Secure Two-party Computation and Communication

by

Vladimir Kolesnikov

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Secure Two-party Computation and Communication

Vladimir Kolesnikov

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2006

In this dissertation, we address several issues that arise in protecting communication between parties, as well as in the area of secure function evaluation. Intuitively, the notion of secure function evaluation is clear and natural: several parties wish to compute some function of their inputs without revealing any information about the inputs, other than what is implied by the value of the function. Research included in this dissertation follows three main directions, briefly described below.

The first direction (Chapters 3 and 4) is the design of efficient protocols for concrete functions of interest. Specifically, we present new, more efficient protocols for securely computing the Greater Than (GT) function on the inputs of two parties. Secure evaluation of GT is frequently needed in financial transactions. We introduce new primitives, which are convenient building blocks for more complex tasks, and generalize our GT solutions to satisfy them. Based on this, we construct secure auction protocols, protocols for determining whether an integer lies on an interval, and others.

The second direction (Chapter 5) is a fundamental approach to secure evaluation of any function, given as a boolean circuit. We present a very efficient information-theoretic (IT) reduction from the problem of secure evaluation of a polysize formula (or, equivalently, a log-depth boolean circuit) to Oblivious Transfer (a fundamental well-researched cryptographic primitive). Our cost of evaluating each gate of the formula is quadratic in its depth, while in previous reductions it was exponential. Our constructions imply efficient one-round protocols for evaluation of polysize formulas on the players' inputs. We extend our solutions to evaluation of polysize circuits, at the cost of having only computational security.

The third direction (Chapter 6) is research on key exchange (KE). In contrast with the previous two directions, here the goal is for two parties to protect their communication against eavesdropping and active interference of an external attacker. KE is a basic procedure, frequently used to establish secure channels between parties. It is a prerequisite to a large number of protocols, including those of the above two directions. We demonstrate a subtle flaw in a previous family of KE protocols and give new KE definitions for the following practical "bank" setting. Here, a server wishes to exchange a key with a client. They have a shared password, and the client carries a "bank card", capable of storing several cryptographic keys. Finally, we present new, more efficient KE protocols for this setting, and prove their security.

Acknowledgements

I would like to start with thanking my family. It was in kindergarten that they saw the future Doctor of Philosophy in me and sent me to primary school. They also took good care of me.

I consider myself very, very fortunate to have studied under the supervision of Ian F. Blake and Charles Rackoff. It has always been a pleasure to talk to them about work, life, or anything else. I am grateful for and impressed by their generosity, brilliance, patience, sense of humor, support, and great advice for all occasions. I want to be like Ian and Charlie when I grow up.

I would like to thank other committee members: Allan Borodin, Stephen Cook, Kumar Murty, Stefan Saroiu, and my external reader, Shai Halevi, for many discussions and to-the-point comments on the writeup of the thesis and one of the papers. I am grateful to other researchers for advice and enlightening conversations that encouraged and influenced my work and career; among them, Bill Aiello, John Byers, Giovanni Di Crescenzo, Marc Fischlin, Péter Gács, Juan Garay, Oded Goldreich, Viktor Nikolaevich Gorbuzov, Stuart Haber, Yuval Ishai, Hugo Krawczyk, Helger Lipmaa, Ivan Platonovich Martynov, Steve Myers, Staszek Radziszowski, Berry Schoenmakers, Vitaly Shmatikov, and many others.

My graduate school was fun, especially during the first two years, when the deadlines were far. I enjoyed life with co-students Alan, Alex, Antonina, Attila, Charlotte, Chris, Cristiana, Dana, Diego, Javid, Josh, Kleoni, Mark, Marlena, Matei, Mike, Natasa, Philipp, Phuong, Roman, Travis, Tristan, Tsuyoshi, Valentine, Wayne, many co-judokas, co-soccer-players, friends, and, of course, people noted in previous paragraphs. In the past year and a half most of enjoyment of life came from Iuliana.

Finally, I am grateful to USIA, Soros, RIT, BU, UofT, OGS, NSERC, and others for fully paying for my education.

Contents

Chapter 1

Introduction

Cryptography (from Greek "secret writing") has emerged as a tool satisfying the need for secret communication. Primitive approaches to this problem (for example, the Caesar's code) have existed for thousands of years. However, only recently, with the development of fast computing devices, has cryptography grown into a structured and mathematical science. The science of secret communications became more formal and rigorous, and, simultaneously, new directions of cryptography appeared and developed. Modern cryptography encompasses much more than the original intent. Examples of new directions include ability to prove possession of certain secret information without revealing it, means of electronic identification and signing, and much more.

The state of modern communications allows easy access to almost any imaginable resource or person. At the same time, the underlying technology often provides no or very weak guarantees. That is, when Alice asks for something (e.g. to have a message m delivered to Bob), it will probably be done. However, this message not only may be lost, it may also be read and, more importantly, modified by an adversary, while it is in transit. While most Internet traffic is of little or no interest to anyone other than its intended recipient, a portion of it serves transactions of value, and requires serious attention to its security. This is why protection against eavesdropping on and interference with the legitimate communication remains perhaps the most commonly used fruit of cryptography. Later in this section, we give a brief high level overview of the goals and methods of securing communication. Chapter 6 contains our contributions to this area of cryptography.

Suppose for now that we've solved the above problem and Alice is fully satisfied that her communication with Bob, Charlie, and others is private and authentic. Is that all she would need? Imagine a situation where Alice participates in a transaction with Bob, but she does not completely trust him. This may happen in many settings where the participants may have conflicting interests, including contract signing, buy/sell transactions, etc. Securing the communication channel can not provide any assurance that Bob does not cheat. Can we protect Alice's (and everyone else's) interests in this setting? A study of *Secure Function Evaluation* (SFE), which began in the 1980's, emerged from the need to not only communicate, but also to *compute* securely. It addresses the problem of providing security against cheating *participants* of the computation. We give more examples and clarify the goals of secure computing later in this chapter. A substantial part of this thesis (Chapters 3–5) discusses in detail our contribution to this area of cryptography. We remark that the term secure *Multi-Party Computation* (MPC) is often used synonymously with SFE in the literature. Sometimes the term MPC is used to emphasize that there are more than two participants in the computation.

1.1 Securing Communication

Consider the following setting. Two honest parties Alice and Bob trust each other, and wish to talk over a public channel. There is an intruder Eve, who has full control of that channel, and who is not an authorised participant of the conversation. Alice and Bob want to ensure security of their conversation. Informally, this means that the following two conditions are satisfied:

- Privacy. Eve does not learn anything from observing the communication (except its length).

- Integrity. If Alice (resp. Bob) receives a message, then this message was sent by Bob (resp. Alice).

We first note that both Alice and Bob must possess some identity-proving credentials to achieve this level of security. Indeed, if at least one of them does not, then nothing prevents Eve from impersonating that person[1]. In general, either what one knows (some secret), what one is (a fingerprint, retina image, etc.), or what one has (a photo ID) identifies this person. In this thesis, we are interested in communication over a network, and only the first type of credential is useful in our setting[2]. Such credential may be a secret shared password, a binary string, established and trusted public/private key pairs, or some combination of them.

It turns out that it is possible to establish a secure conversation between Alice and Bob if they both possess only a relatively short (100–200-bit) random secret string, which is called a *key*. This secure channel can be obtained by using symmetric encryption and message authentication codes (MAC), many constructions of which are believed to be secure. These techniques are well researched and understood, and we do not discuss their use in establishing secure channels in this thesis. However, we stress that a fresh randomly chosen key is necessary for *each* conversation between Alice and Bob. This is the reason why the following trivial solution is unacceptable: Alice and Bob simply use their credential strings as the keys for securing the channel. We discuss the need for a better solution in more detail in the introduction of Chapter 6.

[1]We note that there are settings where it is necessary to ensure authenticity of only one of the parties. Moreover, anonymity of the other party may be essential. In this case, that party need not have/provide credentials.

[2]A change in assumption may allow use of credentials of other types. For example, if we assume that Alice's fingerprint image is secret, it can be used remotely to certify her identity. This is, however, a non-standard and relatively easily violated assumption.

A natural and satisfactory approach is to have Alice and Bob use their credentials to generate a common fresh key each time they want to communicate. This problem, called *Key Exchange* (KE), turns out to be a far from trivial task, due to the potentially exceptionally strong abilities of the intruder Eve. We note that, despite an immense research effort, even definitions of security of KE even in simple settings, have not been agreed upon.

1.1.1 Overview of Our Contributions

In Chapter 6, we discuss in detail our contributions to defining security of and implementing efficient KE protocols in following practical "combined keys" setting. Here, a server (e.g., a bank) wishes to exchange a key with a client. They have a shared password – a relatively short, 3-10 alphanumeric characters, human-memorable string. In addition, the server has large storage and the client carries a "bank card", which stores several long (100-2000-bit) strings. The latter will be keys that are suitable for use in strong cryptography. We assume that the server's keys are securely stored and cannot be stolen. Unfortunately, a similar level of storage security is unreasonable to expect from clients, who are "regular people". We take advantage of the inherent *logistical* differences in how keys are stored by our clients (password in human memory, long key on the card), to achieve more robust security than what is possible by using either type of key alone. Indeed, possession of long keys allows strong security guarantees against an online attacker. However, the card that stores these keys may be (relatively) easily stolen by a physical attacker. On the other hand, passwords may be memorized, need not be stored, and thus can not be stolen. However, the protection against an online attacker one can hope to achieve with passwords is rather weak – passwords can always be guessed with relatively high probability. The only (somewhat satisfactory) protection against guessing attacks is recognizing them and refusing connection after a predetermined number of password failures.

Combining the benefits of both settings allows us to obtain a system, secure against both *types* of attack, and thus suitable for protection of sensitive information. This model is even more appealing due to its wide acceptance – it is natural for us to think of a card and a password, when we do, say, personal banking.

In addition to the introduction and formalization of this setting, we point out a subtle flaw in some instances of the influential KE protocol of Halevi and Krawczyk [58] in a similar setting, and give a new, more efficient protocol. Chapter 6 discusses in detail the setting and our contributions.

1.2 Securing Computation

Intuitively, the notion of secure function evaluation is quite clear and natural. The setting is as follows: there are several parties (sometimes also called players) $P_1, ..., P_n$, perhaps separated by distance, but connected by private communication channels. Each P_i has a private input x_i, which he wants to contribute to help evaluate some function $f(x_1, ..., x_n)$, defined on the players' inputs. At the same time, each player wishes to maintain the privacy of his input to the maximum level possible. Of course, the value of f might reveal some information about some of the inputs. In fact, this is true for most functions of interest. We wish to ensure that no other information is disclosed during the function evaluation procedure.

A moment's thought would reveal the generality and usefulness of this concept and the practical need for its implementation. Even our everyday life and actions could benefit from these techniques, and much more so military, financial and political applications. For example, the voting process can be roughly represented as evaluation of a "voting function" on the inputs held by each individual voter. A secure evaluation of this function would guarantee both correctness of the tally and voter privacy. An online auction is another such application. Here, the bidders and the auctioneer are players who wish to evaluate the "auction function", whose value is equal to the ID of the highest bidder. At the same time, the privacy of bids is desired, and is in fact a requirement in high-stake auctions. Any imaginable interaction between parties that requires (but lacks) a degree of mutual trust or a trusted intermediary would benefit from such cryptographic tools.

Jumping ahead, we note that the general problem of SFE has been solved in many settings. That is, there exist well-defined protocols that allow secure computation of any function, based on certain believed complexity assumptions. However, the general solution is most often too inefficient to be applied in practice. Therefore, a lot of modern research concentrates on specific problems, for which practical protocols are constructed and proven to be secure. One such simple problem of high practical importance is that of evaluating the Greater Than (GT) predicate on the inputs of two parties. Securely determining the greater of two numbers has applications in building secure auctions, data mining systems, and more.

1.2.1 Overview of Our Contributions

In this section, we overview our contributions to both specialized and general SFE.

We consider in detail the problem of SFE of the GT predicate and related functionalities. In addition to the natural need of SFE of GT in auctions, trading, and a variety of other financial transactions, it is also needed in areas such as distributed database mining. (See introductions to Chapters 3 and 4 for more justification.)

As part of our SFE work, we strengthen a popular notion of Oblivious Transfer (OT), and introduce Strong Conditional Oblivious Transfer (SCOT). SCOT is a more convenient (than OT) building block for more complex protocols. We give a new more efficient one-round algorithm for secure evaluation of the GT predicate and cast it as a SCOT. We discuss its applications and extensions.

Our GT protocol requires both parties to "know" their respective input integers, and not just their encryptions. While this requirement is often natural, it becomes a problem in many situations, such as when we want to outsource the computation to a third party. We then give another GT protocol, which does not have this restriction. We give a formalization for the setting with the helping server, which our protocol satisfies. We show how to easily construct auction and sales protocols from our constructions.

We also describe a generic approach to secure function evaluation (SFE). We present a more efficient than previously known ([29, 30, 60, 61, 63]) information-theoretic (IT) reduction from the problem of evaluation of a polysize formula on the players' inputs to OT. Our constructions imply more efficient one-round protocols for evaluation of polysize formulas on the players' inputs. We also introduce Gate Evaluation Secret Sharing (GESS) – a new type of secret sharing, designed for use in SFE with minimal interaction.

1.3 Organization of this Dissertation

We start (Chapter 2) with presenting basic definitions, notation and facts from the literature, as well as accompanying discussions. We present the necessary background information, clarifying various settings in which we design our protocols. In particular, we discuss the semi-honest model and its significance and use. We then discuss the need for a stronger setting – one with malicious adversaries. We briefly outline the capabilities of parties in the two models and mention the automated compilation procedure for obtaining protocols secure in the malicious model from protocols secure in the semi-honest model. This further justifies the interest in solutions in the semi-honest model and clarifies the relationship between the two models. We discuss the need for one-round SFE and for modelling various computational abilities of parties. We give brief historic overviews of the development of the most important notions and primitives we use.

The rest of the dissertation comprises author's published research results [14, 15, 64, 65]. We start with our research contributions to SFE, and then present our KE results. Each of the chapters contains additional relevant background, summary of the contribution, comparison with prior work, and the detailed writeup of the results.

Our first result [14] is presented in Chapter 3. Here we introduce the notion of SCOT, and present a new more efficient one-round SFE of the GT predicate and cast it as a SCOT. We discuss its applications and extensions.

In Chapter 4, we give another GT protocol, suitable for the setting with the helping server. We discuss natural constructions of secure auctions based on our protocol. This chapter contains the results that appeared in [15].

While Chapters 3 and 4 solve specific problems (GT, auctions), Chapter 5 describes a generic approach to secure function evaluation (SFE). The results of this chapter were reported in [64].

In Chapter 6 we switch from discussing SFE to our results in the area of KE, reported in [65]. In this chapter, we introduce and formalize the "combined keys" setting, point out a flaw in some instances of the protocol of Halevi and Krawczyk [58], and give a new, more efficient KE protocol.

We conclude in Chapter 7 with a brief overview and directions for future work.

Chapter 2

Preliminary Discussion, Definitions and Notation

A lot of our work concerns secure two-party computation. Recall, the goal of secure computation is the design of protocols, execution of which leaks no (or negligible) amount of information about the participants' inputs (other than what is implied by the output of the computation). In this chapter, we provide some background information, intuition, justification, and formal definitions necessary for our discussion in Chapters 3 – 5. Our work on KE requires substantially different background, and we delay its discussion until the appropriate section of Chapter 6. We also introduce relevant primitives and notation, and summarize the abbreviations used throughout this dissertation.

2.1 Historical notes on Secure Function Evaluation

We give a very brief historic overview of the development of the field of SFE.

The problem of secure evaluation of a function was first suggested in 1982 by Yao [89]. He discussed the problem (and the solution) of SFE of a small instance of the GT predicate[1]. At that time, the problem of SFE of general functions seemed hard to approach. In 1986, Yao [90] proposed a protocol for general SFE in the setting with two parties. He relied on a specific assumption (that factoring is hard), and made essential use of the oblivious transfer primitive (see Sect. 2.7.3). The development of the "garbled circuit" approach starts with this work. In 1987, Goldreich, Micali and Wigderson [52, 48] considered a more general setting of SFE with an arbitrary number of parties. Additionally, they used zero-knowledge proofs of Goldwasser, Micali and Rackoff [54], and introduced the compilation of the protocols of the semi-honest model into ones secure in the malicious model (see Sect. 2.2.1 and 2.2.2). The above groundbreaking results [89, 90, 52, 48] were presented quite informally, and the corresponding rigorously presented definitions and constructions appeared later. For example, the subsequent works of Beaver, Micali and Rogaway [5], and Rogaway [85] contains formal definitions of SFE and an implicit description of the garbled circuit construction. The first explicit garbled circuit construction with a full proof appeared in 2004 [70]. In 1988, Kilian [63] showed how to perform general SFE based only on oblivious transfer (see Sect. 2.7.3). Other models, e.g., with computationally unbounded players and private channels were considered [13, 27].

2.2 On the models for Secure Function Evaluation

We present a high level discussion of the relevant models in which we consider the problems we solve. We give the necessary formal definitions in Sect. 2.6.

2.2.1 The Semi-honest Model: Intuition and Justification

For much of our work we assume that the adversary is *semi-honest* (sometimes also called *passive*). Intuitively, this means that he exactly follows the protocol specification,

[1]The problem of secure comparison of two integers is now widely known as Yao's two millionaires problem.

yet attempts to learn additional information by analyzing "everything he sees", i.e. his input, randomness, and the transcript of messages received during the execution. (See Sect. 2.6.2 for more formal exposition.) Although the semi-honest adversary is far weaker than the *malicious* (or *active*) one (one who may arbitrarily deviate from the protocol specification), the use and research of the semi-honest model[2] is well justified.

Firstly, protection against only semi-honest adversaries is often sufficient for real-world applications. Indeed, it is often the case that there is a certain mutual trust among the players executing a protocol. At the same time, it is hard to ensure that the view of computation is destroyed after completion of the protocol, even if both parties wish to do this. This is because of the complex structure of the networks, virtual memory and caching mechanisms; almost always information is stored in several places. A trace of an execution secure in the semi-honest model will be of limited help to an adversary who might later hack into a player's computer and obtain this information. Of course, the input of the hacked player will be compromised, but the private information of other players will remain hidden due to the security properties of the (completed) protocols.

Further, it is often the case that reputation is of high importance to businesses and even to private parties. In many scenarios, the payoff of actively cheating is not too high, while the cost of being caught is significantly higher (e.g. destroyed reputation or a chance of legal action). Even though the probability of being caught (e.g., by a random inspection of software or other methods) may be small, this alone may be a sufficient deterrent from active cheating. Semi-honest cheating, however, is often impossible to detect; therefore, protection is needed against semi-honest players.

Finally, the protocol behaviour may be hidden in a large software or hardware system (there exist heuristic methods for obfuscating an execution process), and the cost of amending the behaviour of such systems may be higher than the potential benefit.

While the above discussion justifies certain direct uses of protocols secure in the semi-

[2]The semi-honest model assumes all players are semi-honest.

honest model, research of this model is also important as a stepping stone to achieving fully secure protocols in the malicious model. There are tools (bit commitments, zero-knowledge proofs, and basic protocols allowing players to securely choose random bits) which allow *automatic* compilation of a protocol secure in the semi-honest model into a protocol computing the same functionality securely in the malicious model. Intuitively, players first commit to their input and securely choose their random tapes. Then, for every message sent by the semi-honest protocol, a zero-knowledge proof is added, convincing other player(s) that the message is formed properly (i.e. consistently with the protocol, player's randomness and previous messages). With a negligible probability of proving a false statement, parties are forced to follow the protocol, and thus their cheating capabilities lie in the semi-honest model.

Further, different players may have different levels of trust. For example, in an auction system, a bidder may trust an established auction house to act semi-honestly. At the same time, neither he, nor the auction house might have such trust in other bidders, thus requiring protection against malicious bidders. In such cases, it is often most efficient to design semi-honest protocols, and then selectively add efficient protection against certain malicious behaviours of certain players. This is the approach we take in our protocols in Chapter 4.

2.2.2 On the Malicious Model

In our work, we don't consider the malicious model in its full generality. We note, however, that we consider some malicious behaviours of SFE participants in Chapter 4. Further, in our discussion of KE, we protect against certain adversarial behaviours, relevant to our specific problem. In particular, our adversary not only controls some of the players of the protocol, but also is a powerful external entity, able to arbitrarily schedule and execute multiple instances of the KE protocol between honest and malicious players. On the other hand, we make certain assumptions on the inputs of honest parties (e.g. that they are randomly chosen equal strings that are not even partially known to the adversary). Defining security in this (class of) settings is a challenging task of modern cryptographic research. Indeed, the main contribution of Chapter 6 is a key exchange definition.

We stress that security definitions of KE are significantly different from those of SFE in the malicious model. Therefore, we do not include the latter in the discussion of this chapter, other than briefly mentioning them where appropriate.

2.3 One Round SFE

SFE constructions presented in this dissertation are one round. That is, the *initiator* Alice sends the first message to the other party Bob; Bob replies (he is the *responder*), and Alice computes $f(x, y)$ (thus Alice is also the *receiver*). We now briefly justify the research effort in seeking one round solutions.

One round SFE is particularly interesting for several reasons.

Firstly, from a practical point of view, interaction necessarily involves latencies in message deliveries, and in many practical situations waiting for messages dominates the entire computation time.

Secondly, a large volume of research, e.g. [22, 40, 60, 61], aims specifically at reducing round complexity of multi-party protocols. (We note that polynomial time one-round SFE protocols are known in many, but not all, settings.) Further investigation of the two-party one-round model may help increase our understanding of general secure multi-party computation.

Thirdly, non-interactivity is desired or even necessary in many real-life settings. For example, the recently popular area of secure autonomous agent computing (see, e.g. [2, 22]) relies on one round protocols, commonly implemented via encrypted circuit constructions. One example, discussed in [2], is that of a shopping agent that would accept a sales offer if it is below a certain threshold. Auctions and other financial applications also benefit from one round solutions, eliminating or reducing the need for online waiting and synchronization between parties.

Finally, and, perhaps, most importantly, one round solutions have the following unique opportunity for performance improvement. Consider the practical setting, where the players have unequal levels of trust. For example, Bob might be a bank, and Alice might be his customer. As discussed in Sect. 2.2.1, Bob might value his reputation highly, and will not risk active cheating. Alice, on the other hand, may have less to lose, and might be tempted to deviate from prescribed protocols to gain potential benefit.

In such situations, where Alice is assumed to be malicious, and Bob is assumed to be semi-honest, one round protocols limit Alice's abilities to cheat. Indeed, all that Alice does is encode her input and decode the output. We note that in many situations it is relatively easy to ensure that Alice either cannot cheat while performing these actions, or that she cannot benefit from cheating. While, in general, complex and expensive zero-knowledge proof techniques can be used, light-weight solutions are applicable in this setting. While Alice cannot be prevented from substituting her input with another allowed value, it can often be ensured that she does not learn anything from protocol executions where she sends Bob an invalid input or its improper encoding. Conditional Disclosure of Secrets (CDS) techniques of Aiello, Ishai and Reingold [1] and Laur and Lipmaa [68] serve this purpose efficiently. Further, Bob can send Alice a random and secret encoding of the output, instead of the plaintext output. In this case, Alice would not be able to lie about the value she received. Effectively, one-round protocols allow us to shift computation to Bob, where it is cheaper, due to our semi-honest level of trust in him. See Sect. 4.5.1 for more discussion.

2.4 On Computational Abilities of Parties

It is standard in cryptography to model adversaries as Probabilistic Polynomial Time (PPT) Turing machines. This is a reasonable assumption, and the achieved security satisfies the needs of most of today's applications. However, for the sake of theoretical interest and several practical reasons, it is often beneficial to consider a stronger computationally unbounded adversary. Below we briefly outline these reasons.

Computational security relies on the assumption of hardness of solving certain problems. Once a protocol is executed, the corresponding concrete instances of hard problems are fixed. A solution of any of these problems at any time in the future may reveal entire private inputs of the parties. Thus, a participant of the protocol or even a communication channel observer may record the transcript of the protocol execution and attempt to recover information years later. We stress that the success of this attack may be quite likely due to the exponentially decreasing costs of processing power and due to possible algorithmic breakthroughs in solving underlying hard problems. While in some settings, it is sufficient to keep the privacy of the input only for a few years, many applications (e.g. political, military, even financial) may require protection for indefinite periods of time. For those applications, it often helps to consider computationally unbounded adversaries.

Further, in many settings, the execution of protocols is not performed in isolation. That is, a particular protocol may be executed sequentially, in parallel, or concurrently with another instance of itself or other protocols. Computational security ensures that the information leaked during the execution of a single instance of a protocol does not help the adversary. However, no security guarantees can be inferred from this for the setting when protocols are executed concurrently, and a complicated special consideration is necessary for a particular protocol. At the same time, in most cases a composition of protocols secure against a computationally unbounded adversary is secure. See [67] for more discussion and counterexamples.

It is not hard to see that in the two-party computation over standard communication channels, it is not possible to achieve security if both parties are computationally unbounded. Therefore, it is interesting to consider settings, where one of the parties (either Alice or Bob) is polytime, while the other is not. We still get the benefits described above, with respect to the corresponding party. Note that in practice, the input of only one of the parties (say, Bob) may need to be protected indefinitely. Then we would benefit from a protocol which is secure against computationally unbounded Alice. We further mention, but do not address in this dissertation, that the assumptions on parties' abilities may be fine-tuned. For example, it is reasonable to assume that Alice is a PPT Turing machine during the execution of the protocol, but has unlimited time to analyze the transcript, once the protocol has completed.

Jumping ahead, we note that in this dissertation (Chapters 3 and 4), we approach one-round SFE of concrete problems in the setting with computationally unbounded receiver Alice. We note that many of the current one round SFE protocols in the setting with unbounded Alice are not very efficient. The best currently known general approach [30, 63, 86] is quite inefficient and only works for NC^1 circuits. At the same time, if Alice is assumed to be polytime bounded, we could use the very efficient Yao's garbled circuit approach [70, 76, 85, 89] at a cost linear with the size of the circuit. We remark that our solutions are in the more difficult setting (unbounded Alice), while achieving performance only slightly worse than the best known approach in the easier (polytime bounded Alice) setting.

2.5 Notation and Abbreviations

Notation 1. *For strings of the same length $s, t \in \{0,1\}^*$, let $s \oplus t$ denote their bitwise exclusive-or.*

Notation 2. *We denote uniformly random sampling by the $\in_R$ operator. For example, $r \in_R D$ means "choose r uniformly at random from D".*

Throughout this dissertation, we will often refer to (concrete or abstract) adversaries as Adv.

For reference, we present the list of most common abbreviations used in this thesis. The reader might skip it at the first reading, and refer to it if necessary.

BP – Branching Program

CEM – Conditional Encrypted Mapping

COT – Conditional Oblivious Transfer

GESS – Gate Evaluation Secret Sharing

GT – Greater Than

IT – Information Theoretic

KE – Key Exchange

MAC – Message Authentication Code

OT – Oblivious Transfer

PPT – Probabilistic Polynomial Time

PSPP – Private Selective Payments Protocol

PRFG – Pseudorandom Function Generator

SCOT – Strong Conditional Oblivious Transfer

SFE – Secure Function Evaluation

Throughout this dissertation we refer to parties by names (Alice, Bob), their function (Sender, Receiver, Server, Client), index (P_i), or the initial of the name or function (A, B, S, R). In our notation, Alice is the party who sends the first message of the protocol to Bob. The reason for using these naming conventions is that it appears to be clearer and more convenient to refer to players differently in the contexts of each particular problem we are considering.

2.6 Formal Background: Definitions

We present standard definitions from the literature which are necessary to introduce the subject of the dissertation.

Definition 1. *A function $f : \mathbb{N} \mapsto [0,1]$ is called* negligible *if for every positive polynomial p and all sufficiently large k, $f(k) < 1/p(k)$.*

A function $f : \mathbb{N} \mapsto [0,1]$ is called overwhelming*, if the function $1 - f$ is negligible.*

Most of the modern theoretical cryptography is built around probabilities, random distributions and their relations. The following definition formalizes the basic notions relating to probability ensembles.

Definition 2. *A* probability ensemble indexed by $S \subseteq \{0,1\}^*$ *is a family $\{X_w\}_{w \in S}$, so that each X_w is a random variable (or distribution) which ranges over (a subset of) $\{0,1\}^{poly(|w|)}$.*

Statistical distance between distributions X_w and Y_w is defined as $Dist(X_w, Y_w) = 1/2 \sum_{\alpha} | \ Pr[X_w = \alpha] - \ Pr[Y_w = \alpha]|$.

We say that two such ensembles, $X = \{X_w\}_{w \in S}$ and $Y = \{Y_w\}_{w \in S}$ are identically distributed *(and write $X \equiv Y$), if $\forall w \in S, Dist(X_w, Y_w) = 0$*

Ensembles X and Y are said to be statistically indistinguishable*, if for some negligible function $\mu : \mathbb{N} \mapsto [0,1]$ and all $w \in S$, $Dist(X_w, Y_w) < \mu(|w|)$. In such case we write $X \overset{s}{\equiv} Y$.*

We say that ensembles X and Y are computationally indistinguishable*, if for every family of (possibly non-uniform) polynomial-size circuits, $\{D_n\}_{n \in \mathbb{N}}$ (distinguishers), there* *exists a negligible function* $\mu : \mathbb{N} \mapsto [0,1]$, *such that* $\forall w \in S, | \, Pr[D_{|w|}(w, X_w) = 1] - Pr[D_{|w|}(w, Y_w) = 1]| < \mu(|w|)$. *In such case we write* $X \overset{c}{\equiv} Y$.

2.6.1 General Secure Two-Party Computation

A two-party *functionality* is a possibly random process that maps two inputs (one input per party) to two outputs (one per each party). We wish to evaluate functionalities (rather than functions), since the participants of a protocol have separate inputs, and, in general, wish to compute different functions of the common input and randomness. In this dissertation, we will use both terms (function and functionality) interchangeably, to mean the more general notion.

Intuitively, a protocol is a formal prescription, or a program, of actions of parties who interact with each other by sending messages. The output of the protocol is the output of all its participants.

The problem of secure two-party computation is to devise a protocol that securely computes a given *functionality* $f : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^* \times \{0,1\}^*$. We will be interested in both deterministic and general (randomized) functionalities.

Definition 3. *We say that a protocol* Π computes *a general functionality* f *if the output distribution of the protocol on input* $\vec{x}$ *is distributed statistically close to* $f(\vec{x})$.

The definition of security of computation in the semi-honest model is discussed below.

2.6.2 Definition of Security of SFE in the Semi-honest Model

Recall, a semi-honest party is a party who follows the protocol, but keeps a record of everything he sees. In particular, he tosses fair coins and sends messages to the other party according to the protocol. In the end he tries to compute a function of the other party's input. Notice that it is sufficient for the semi-honest party to keep a record only of internal coin tosses and the received messages. Definitions presented in this section are standard in the literature, and appear, for example, in Goldreich [49].

First, to formalize the notion of keeping a record, define the "view" of the computation.

Definition 4. *(view of the two-party computation)*

Let $f : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^* \times \{0,1\}^*$ *be a functionality where* $f_1(x,y)$ *denotes the first element of* $f(x,y)$*, and* Π *be a two-party protocol for computing* f*.*

The view *of the first party during the execution of* Π *on* (x,y)*, denoted* $VIEW_1^{\Pi}(x,y)$*, is* $(x, r, m_1, ..., m_t)$*, where* r *represents the output of the first party's internal coin tosses, and* m_i *represents the* i^{th} *message it has received.*

The output *of the first party during an execution of* Π *on* (x,y)*, denoted* $OUTPUT_1^{\Pi}(x,y)$*, is determined by the party's view of the execution.*

The view and output of the second party is defined analogously.

We note that $\mathrm{VIEW}_P^{\Pi}(x,y)$ is a random variable over the random coins of the parties.

We now proceed and define privacy for the semi-honest model. A common approach to this is to use the simulation paradigm. We say that a protocol securely computes functionality f, if whatever can be computed (by either party) from the execution of the protocol, can be essentially obtained from that party's input and the output of the computation. Notice that it is sufficient to (efficiently) "simulate the view" of each (semi-honest) party, since anything that can be obtained from participation in the protocol, can be obtained from the view of the party.

Definition 5. *(privacy w.r.t. semi-honest behavior) For a* deterministic *functionality* f*, we say that a protocol* Π securely computes f *if there exist probabilistic polytime algorithms* S_1 *and* S_2 *(simulators), such that*

$$\{S_1(x, f_1(x,y))\}_{x,y\in\{0,1\}^*} \overset{c}{\equiv} \{VIEW_1^{\Pi}(x,y)\}_{x,y\in\{0,1\}^*}$$

$$\{S_2(y, f_2(x,y))\}_{x,y\in\{0,1\}^*} \overset{c}{\equiv} \{VIEW_2^{\Pi}(x,y)\}_{x,y\in\{0,1\}^*}$$

where $|x| = |y|$.

We say that Π *privately computes a general functionality* f*, if there exist probabilistic polytime algorithms* S_1 *and* S_2 *(simulators), such that*

$$\{S_1(x, f_1(x,y)), f_2(x,y)\}_{x,y\in\{0,1\}^*} \overset{c}{\equiv} \{VIEW_1^{\Pi}(x,y), OUTPUT_2^{\Pi}(x,y)\}_{x,y\in\{0,1\}^*}$$

$$\{f_1(x,y), S_2(y, f_2(x,y))\}_{x,y\in\{0,1\}^*} \overset{c}{\equiv} \{OUTPUT_1^{\Pi}(x,y), VIEW_2^{\Pi}(x,y)\}_{x,y\in\{0,1\}^*}$$

Note that the above $VIEW_1^{\Pi}(x,y)$, $VIEW_2^{\Pi}(x,y)$, $OUTPUT_1^{\Pi}(x,y)$ *and* $OUTPUT_2^{\Pi}(x,y)$ *are related random variables, defined as a function of the same random execution.*

We note that for the ease of understanding, some of the constructions and analysis in this dissertation are presented with respect to fixed parameters. We stress that we have in mind their asymptotic notions. Therefore, for example, when talking about a view of a party $\mathrm{VIEW}_P^{\Pi}(x,y)$, we mean an appropriately indexed ensemble $\{\mathrm{VIEW}_P^{\Pi}(x,y)\}$ of views.

2.7 Employed Cryptographic Primitives

2.7.1 Public-Key Encryption

Public-key encryption is one of the most important discoveries in the area of cryptography. Its main idea is to separate the keys needed for encryption and decryption of messages. The key for encryption could be public, e.g., for the purpose of allowing anyone to securely send a message to the recipient. The decryption key cannot be efficiently obtained from the public key, and is kept secret.

Diffie and Hellman are widely credited as the inventors of public key cryptography for their seminal contribution [34]. Soon after the appearance of [34], Rivest, Shamir and Adelman [83, 84] introduced the well-known RSA encryption scheme. The first invention of public key encryption is sometimes credited to J. H. Ellis and Clifford Cocks, who showed the feasibility of the concept and proposed an RSA-like scheme, respectively. This work was done in in the early 1970s. See [36] for a historical overview. Ellis and Cocks were staff members at GCHQ (a British intelligence agency), and the fact of their inventions was kept secret until 1997.

Definition 6. *(Public Key Encryption Scheme) Let $E = (Gen, Enc, Dec)$ be a triple of PPT algorithms, where:*

- *$Gen : 1^k \mapsto PK \times SK$ is the key generation algorithm. On input 1^k, Gen outputs public and private (secret) keys $pk \in PK, sk \in SK$.*
- *$Enc : PK \times M \mapsto C$ is the encryption algorithm, where M and C are the plaintext and ciphertext domains.*
- *$Dec : SK \times C \mapsto M \cup \perp$ is the decryption algorithm. We intend that Dec may output $\perp$ if the ciphertext is invalid.*

We say that E is a public key encryption scheme*, if it satisfies the correctness property: $\forall m \in M, \forall (pk, sk)$ generated by Gen, $Dec_{sk}(Enc_{pk}(m)) = m$.*

We write Enc_{pk} and Dec_{sk} to emphasize the use of the corresponding key. We may sometimes abuse the notation and omit the subscript where the meaning is clear from the context. We also note that domains M and C are dependent on the public key pk.

Note that Def. 6 refers to the functionality of encryption but not to its security. In particular, identity transformations Enc, Dec satisfy our requirements. In practice, of course, we wish to have some guarantee of privacy of the encrypted data. In this dissertation, we make use of two different notions of security of encryption – *semantic* (or, equivalently, *chosen plaintext attack* (CPA)) security and *chosen ciphertext attack* (CCA)) security. We note that the latter is often referred in the literature as *chosen ciphertext attack - 2, or CCA2.* We define these notions below.

Semantic Security

A semantically secure cryptosystem satisfies our intuitive notion that no new knowledge about the message is gained by a polytime observer from seeing an encryption. The formalization of this notion and a construction based on the quadratic residuocity assumption is due to Goldwasser and Micali [53].

Consider the following game G_{CPA} an adversary Adv plays. $Gen(1^k)$ is run, and Adv is given the generated pk. Adversary then chooses two messages $m_1, m_2 \in M$. A bit b is randomly chosen and Adv is given the challenge $c = Enc_{pk}(m_b)$. Adv then outputs a bit b'. Adv wins if $b = b'$.

Definition 7. *We say that an encryption scheme E is* semantically secure*, if for every polytime (in k) Adv, his success in winning in G_{CPA} is negligibly different from* $1/2$.

We will use *probabilistic* encryption schemes. Informally, a scheme is probabilistic (or *randomized*), if its encryption function uses randomness to encrypt a plaintext as one of many possible ciphertexts. We note that a semantically secure scheme must be probabilistic. Indeed, in particular, a polytime Adv must not be able to relate two encryptions of the same message, which is easily done for non-probabilistic schemes (otherwise it is easy to construct an Adv who wins G_{CPA}). We will use the property of unlinkability of encryptions.

We say that an encryption scheme allows *re-randomization* if a random encryption of a plaintext can be (efficiently) computed from a corresponding ciphertext and the public key.

Chosen Ciphertext Security

We note that in many applications semantic security is insufficient. In particular, if an adversary is given an encryption of a message m, he may be able to construct encryptions of messages related to m. (Encryption schemes that allow that are called *malleable.*) The following notion of security denies Adv this power (actually, it provides a slightly stronger guarantee than non-malleability).

Consider the following game G_{CCA}. It proceeds exactly as G_{CPA}, with the exception that Adv is given the following additional power. Adv is allowed to obtain decryptions of ciphertexts $c' \neq c$ of his choice (by querying the decryption oracle $O(c') = Dec_{sk}(c')$) whenever he wishes. We note that, in particular, Adv is allowed to query $O(c')$ before he chooses the messages m_1, m_2.

Definition 8. *We say that an encryption scheme E is* chosen ciphertext (CCA) secure*, if for every polytime (in k) Adv, his success in winning in G_{CCA} is negligibly different from* $1/2$.

Intuitively, it is clear that Adv should not be able to "malleate" encryptions; if he could, then he could query Dec_{sk} on encryptions related to m_b, and thus gain advantage in G_{CCA}. We use CCA-secure encryption schemes in construction of KE protocols.

This important notion was introduced by Rackoff and Simon [82]. A weaker and less natural notion of CCA1 security was earlier proposed by Naor and Yung [72]. The CCA1 adversary has the power similar to the CCA one, except that he is not allowed to access the decryption oracle after he is presented with the challenge. Dolev, Dwork and Naor [35] later introduced the notion of non-malleability and gave proof-of-concept constructions of non-malleable encryption, string commitment and zero-knowledge proof schemes. These constructions rely on non-interactive zero-knowledge proofs, and require a large common random string. Today, CCA-secure schemes are practical. The most popular and efficient Cramer-Shoup scheme [31] relies on Decisional Diffie-Hellman assumption and requires only a few exponentiations in a group.

2.7.2 Homomorphic Encryption

We say that a public-key encryption scheme E is *homomorphic*, if for some operations $\oplus$ and $\otimes$ (defined on possibly different domains), it holds that for all key pairs (pk, sk), $x \oplus y = Dec_{sk}(Enc_{pk}(x) \otimes Enc_{pk}(y))$. A scheme is called additively (multiplicatively) homomorphic if it is homomorphic with respect to the corresponding operation (e.g. additive scheme allows to compute $Enc_{pk}(x + y)$ from $Enc_{pk}(x)$ and $Enc_{pk}(y)$). Many of the commonly used schemes are homomorphic. For example, the ElGamal scheme [45] is multiplicatively homomorphic, and Goldwasser-Micali [53] and Paillier [78] schemes are additively homomorphic. Unfortunately, it is not known whether there exists a scheme that is algebraically (i.e. both additively and multiplicatively) homomorphic. Moreover, the existence of such schemes seems unlikely. We note that an additively homomorphic scheme allows multiplication by a known constant, i.e. computing $Enc_{pk}(cx)$ from $Enc_{pk}(x)$ and c, via repeated addition.

The Paillier Encryption Scheme

Our protocols (in Chapters 3 and 4) require an additional property of the encryption scheme – large plaintext size, or *bandwidth*. The Paillier scheme [78] satisfies all our requirements, and we will instantiate our GT protocols with it. We present the scheme, but omit the number-theoretic justification.

Key generation: Let N be an RSA modulus $N = pq$, where p, q are large primes. Let g be an integer of order $N\alpha$ in $\mathbb{Z}^*_{N^2}$, for some integer α. The public key $pk = (N, g)$ and the secret key $sk = \lambda(N) = \mathrm{lcm}((p-1),(q-1))$, where $\lambda(N)$ is Carmichael's lambda function.

Encryption: to encrypt $m \in \mathbb{Z}_N$, compute $\mathrm{Enc}(m) = g^m r^N \mod N^2$, where $r \in_R \mathbb{Z}^*_N$.

Decryption: to decrypt a ciphertext c, compute $m = \frac{L(c^{\lambda(N)} \mod N^2)}{L(g^{\lambda(N)} \mod N^2)} \mod N$, where

$L(u) = \frac{u-1}{N}$ takes as input an element from the set $S_N = \{u < N^2 | u = 1 \mod N\}$.

Re-randomization: to re-randomize a ciphertext c, multiply it by a random encryption of 0, i.e. compute $cr^N \mod N^2$, for $r \in_R \mathbb{Z}_N^*$.

The underlying security assumption is that the so-called composite residuocity class problem is intractable. This assumption is referred to as the Computational Composite Residuocity Assumption (CCRA). It is potentially stronger than the RSA assumption, as well as the quadratic residuocity assumption. We refer the interested reader to [78] for further details.

2.7.3 Oblivious Transfer (OT)

Oblivious Transfer is a basic and very powerful two-party functionality. A secure evaluation of OT (often simply called OT) is an execution of a protocol between two parties, Sender S and Receiver R. R has an single bit b as input; S has two input strings s_0, s_1. In the end of this execution, R learns the secret of his choice, s_b, and nothing else. S learns nothing. More formally, OT is a secure evaluation of the following functionality

Functionality 1.

$$f_{\mathrm{OT}}(b, (s_0, s_1)) = (s_b, \text{empty string})$$

Ever since its introduction by Rabin [81], OT has been a subject of a large amount of research. A number of variants and reformulations of OT, as well as reductions among those variants have been proposed over the years. See Sect. 4.2 for more discussion of some OT variants. Efficient OT constructions exist based on specific assumptions, such as hardness of factoring or the Diffie-Hellman assumption [81, 9, 75, 1]. OT can be built from generic assumptions, such as enhanced trapdoor permutations [37, 49, 56], and from physical assumptions, such as channel noise.

We note that OT is an especially important primitive, since the problem of secure computation of any function can be reduced to OT. That is, there exist unconditional (as well as more efficient, but conditional on other hardness assumptions) SFE constructions which use OT as a building block. The first unconditional reduction is due to Kilian [63]; our work (Chapter 5) is an efficiency improvement of known reductions.

We note that a natural extension of OT, where one of n (as opposed to one of 2) secrets is transferred to the receiver, is also considered in the literature. These variants are often called 1-out of-n OT.

2.7.4 Pseudo-random Function Generator (PRFG)

Intuitively, a function generator is a procedure that takes a k-bit seed s, and produces a function $F_s : \{0,1\}^k \mapsto \{0,1\}^k$. Of course, F_s is an exponentially large object; rather than actually outputting it, we merely require that the generator can efficiently evaluate F_s on any input. Informally, a function generator is pseudorandom if no efficient algorithm can tell the difference between being given a blackbox for a random function, and being given a black box for F_s for a randomly chosen seed s. The notion of PRFG and the construction of PRFG from a pseudorandom number generator were proposed by Goldreich, Goldwasser and Micali [50].

Definition 9. *A* Function Generator *F associates with each k and each $s \in \{0,1\}^k$ a function $F_s : \{0,1\}^k \mapsto \{0,1\}^k$, such that there is a polynomial time algorithm that given $s \in \{0,1\}^k$ and $x \in \{0,1\}^k$, computes $F_s(x)$.*

Such F is a pseudorandom *function generator (PRFG), if for every polynomial time distinguisher with oracle access D^f the following holds.*

Let seed $s \in_R \{0,1\}^k$ and function $R \in_R \{0,1\}^k \mapsto \{0,1\}^k$ are chosen randomly. Then $|Prob(D^{F_s}(1^k) = 1) - Prob(D^R(1^k) = 1)| < 1/k^c$ for every c and sufficiently large k.

We remark that in practice, PRFGs are not defined for all k, but rather, for some fixed sufficiently large k, thus (technically, but not in intent or spirit) violating the requirements of Def. 9. DES and AES are such examples. It is widely believed that it is infeasible to distinguish the output of a random element of the AES family from a randomly chosen function on the same domain.

2.7.5 Message Authentication Code (MAC)

MAC is a tool for ensuring authenticity of messages. It is most commonly used in authenticating communication. (We note that in this work we use MAC for this purpose as well.) In this setting, two parties have shared a random private key ℓ of length k; later, one of them wants to use ℓ to authenticate messages by generating corresponding tags. The tag generation function is stateless and deterministic, and verification is done by applying the tagging function to compute the correct tag of the given message, and comparing it with the candidate tag. More formally:

Definition 10. *A* Message Authentication Code (MAC) *is a stateless deterministic algorithm $MAC : \{0,1\}^k \times \{0,1\}^* \mapsto TAG$. On input key $\ell \in \{0,1\}^k$ and a message $m \in \{0,1\}^*$, MAC outputs a tag $\tau \in TAG$. (Here TAG is the domain of tags, which depends on k, and is independent of the signed message length.) We will sometimes write $MAC_\ell(m)$ to mean $MAC(\ell, m)$.*

Let $\ell \in_R \{0,1\}^k$. Let Adv be a polytime adversary with an access to the MAC oracle $O(m) = MAC_\ell(m)$. Adv outputs a message m' and its alleged signature τ', and must never call $O(m')$. We say that MAC is secure, if for every such Adv, $Prob(\tau' = MAC_\ell(m')) < 1/k^c$ for every c and sufficiently large k.

We remark that in practice, MAC schemes are built directly from PRFGs. Like PRFGs, practical MAC schemes are not defined for all k, but rather, for some fixed sufficiently large k.

We note that MAC is a special case of the more general notion of message authentication schemes. MAC satisfies the strongest requirements of message authentication schemes [8], and is sufficient for our purposes.

Chapter 3

Secure Evaluation of the Greater Than Predicate

3.1 Introduction

Summary of the contributions of the chapter. We consider the problem of securely computing the Greater Than (GT) predicate and its generalization – securely determining membership in a union of intervals. We approach these problems from the point of view of Q-Conditional Oblivious Transfer (Q-COT), introduced by Di Crescenzo, Ostrovsky and Rajagopalan [32]. Q-COT is an oblivious transfer that occurs *iff* predicate Q evaluates to `true` on the parties' inputs. We are working in the semi-honest model with *computationally unbounded* receiver.

We propose: (i) a stronger, simple and intuitive definition of COT, which we call *strong* COT, or Q-SCOT. (ii) A simpler and more efficient one-round protocol for securely computing GT and GT-SCOT. (iii) A simple and efficient modular construction reducing SCOT based on membership in a union of intervals (which we call UI-SCOT) to GT-SCOT, producing an efficient one-round UI-SCOT.

3.1.1 Motivation of the Problem and the Setting

The work presented in this chapter falls into the area of constructing efficient secure multi-party protocols for interesting functionalities. The more basic the functionality, the more common is its use, and the more significant is the value of any performance improvement of the corresponding protocol. We start with presenting the problems we investigate and their motivation.

The base functionality we consider – Greater Than (GT) – is one of the most basic and commonly used. Secure evaluation of GT is also one of the most famous and well-researched problems in cryptography. There exist a vast number of applications relying on it, such as auction systems or price negotiations.

Another typical example would be secure distributed database mining. The setting is as follows: several parties, each having a private database, wish to determine some properties of, or perform computations on, their *joint* database. The need for such computations commonly arises in the modern business world. A typical example would be two credit card companies wishing to analyze their joint client databases for fraud fighting purposes, but unwilling to disclose to the competitor anything that is not absolutely necessary. Many interesting properties and computations, such as transaction classification or rule mining, heavily involve evaluating a large number of instances of GT [62, 69]. Because of the large size of the databases, even a minor efficiency gain in computing GT results in significant performance improvements and cost savings.

Consider the problem of determining whether a point belongs to the union of a set of intervals. (Sometimes we will refer to it as the membership in a set of intervals.) We note that functionalities such as this are less studied, but nevertheless are very useful. Their immediate uses lie in appointment scheduling, flexible timestamp verification, expression evaluation, in the areas of computational geometry, biometrics, and many others. Certain kinds of set membership problems, as studied by Freedman, Nissim and Pinkas [43], can be represented succinctly as instances of problems we consider. For example, the problem of membership in a set consisting of all even integers on a large interval (y, z) can be represented as a conjunction of two small instances of interval memberships ($S = \{x | x_0 < 1 \wedge x \in (y, z)\}$, where x_0 is the low bit of x). In such cases, using our solutions may have significant advantages over the general set intersection solution of [43].

We consider the setting with computationally unbounded receiver (Alice) and poly-time limited Bob. We refer the reader to Sect. 2.4 for the justification of this model, more details and discussion. We note that our protocols are secure in the more difficult setting (unbounded Alice), while achieving performance only slightly worse than the best known approach in the easier (polynomially bounded Alice) setting.

3.1.2 Contributions and Outline this Chapter

We start with a discussion of Conditional Oblivious Transfer (COT) (Sect. 3.2). We wish to strengthen the current definition of [32] in several respects. Firstly, we observe that the definition of [32] does not require the privacy of the sender's private input. Secondly, we propose and justify the "1-out-of-2" Q-COT, where the receiver obtains one of two possible secret messages depending on Q, but without learning the value of Q. This is opposed to the "all-or-nothing" approach of [32] where the receiver receives either a message or nothing, which necessarily reveals the value of Q. Our approach significantly adds to the flexibility of COT functionalities and allows for more powerful compositions of COT protocols. We propose a definition of *strong* conditional oblivious transfer (SCOT) that incorporates the above observations and some other (minor) points. (We note that DiCrescenzo [33] informally described a concept of Symmetrically-private COT, which is similar to SCOT. We stress that our work was performed independently of [33]. Unlike [33], we give formal definitions of SCOT; our discussions and justifications further contribute to the understanding of the concept. We also note that other notions similar to COT were previously proposed, some of which are briefly discussed in Sect. 4.2. However, the notion of COT is most relevant to our discussion, and we limit our comparisons to this notion only.)

Then, in Sect. 3.3, we discuss previous work on the GT problem and present our main tool – an efficient protocol for computing GT-SCOT built from a homomorphic encryption scheme. We exploit the structure of the GT predicate in a novel way to arrive at a solution that is more efficient and flexible than the best previously known (of Fischlin [41] and DiCrescenzo [33]) for our model with unbounded Alice. Additionally, our construction is the first to offer transfer of c-bit secrets, with $c \approx 1000$ for practical applications, at no extra cost, with *one* invocation of the protocol, as opposed to the necessary c invocations of previous protocols. This results in additional significant efficiency gains.

Then, in Sect. 3.4, we show how to use the bandwidth of our GT-COT solution and present protocols for efficiently computing SCOT based on the interval membership (which we call I-SCOT) and SCOT based on the membership in a union of k intervals (which we call k-UI-SCOT). Because of their modularity, these protocols can also be constructed based on Fischlin's [41] and Di Crescenzo's [33] solutions at the efficiency loss described in the previous paragraph. Because they leak the private inputs of the sender, we do not know of an efficient way to extend solutions of [32] to compute these functionalities. We remark on how to use UI-SCOT to compute the conjunction or disjunction of the memberships in unions of intervals. Finally, we compare and summarize resource requirements of schemes of Fischlin, Di Crescenzo, Di Crescenzo et al., and ours in the Table in Sect. 3.4.2.

3.1.3 Our Setting

We remind the reader what our setting is. We are working in a setting with two semi-honest participants, who use randomness in their computation. A computationally unbounded Alice sends the first message; polytime limited Bob replies, and Alice outputs the value of the evaluated function. We prove security with respect to standard definitions. See Goldreich [49] or Chapter 2 (specifically Sect. 2.6.2) for definitions and in-depth discussion.

As discussed in Sect. 2.6.2, although our constructions and analysis are presented for fixed security and correctness[1] parameters ν and λ, we have in mind their asymptotic notions.

3.2 Strong Conditional Oblivious Transfer

The notion of COT was introduced by Di Crescenzo, Ostrovsky and Rajagopalan [32] in the context of timed-release encryption. It is a variant of Oblivious Transfer (OT) introduced by Rabin [81]. Intuitively, in COT, the two participants, a receiver R and a sender S, have private inputs x and y respectively, and share a public predicate $Q(\cdot,\cdot)$. S has a secret s he wishes (obliviously to himself) to transfer to R iff $Q(x,y)=1$. If $Q(x,y)=0$, no information about s is transferred to R. R's private input and the value of the predicate remain computationally hidden from S.

3.2.1 Our Definitions

We start by describing several ways of strengthening the existing definition with the goal of increasing modularity and widening the applicability of SCOT protocols. Our own construction for UI-SCOT, for example, requires its building blocks to have the proposed features.

First, while sufficient for the proposed timed-release encryption scheme, the definition of [32] lacks the requirement of secrecy of the sender's private input. We would like the new definition to include this requirement.

Secondly, we prefer the "1-out-of-2" approach. In our proposed setting, the sender possesses two secrets s_0 and s_1, and wishes (obliviously to himself) to send s_1 if $Q(x,y) =$

[1]Correctness parameter specifies the allowed probability of error in the protocols.

1, and to send s_0 otherwise. Unlike the COT "all-or-nothing" definition, this allows SCOT protocols to have the property of *not revealing* $Q(x, y)$ *to the receiver.* This proposal strengthens the definition since while a SCOT protocol can be trivially modified to satisfy COT definitions of [32], it is not hard to see that the opposite does not (efficiently) hold[2]. Further, note that it follows from our requirements that a Q-SCOT protocol can be trivially modified into a $(\neg Q)$-SCOT protocol. This also does not hold for COT. We will use this important property in our constructions later in this chapter.

The definition of [32] is presented in the common random string model, while ours is not. We do not see the need of including it, since we are able to construct useful (more) efficient COT protocols in the standard setting.

Finally, as a minor point, we only require statistical, as opposed to perfect, correctness and security against R, to allow for easier analysis of the protocols and wider applicability of the SCOT notion.

We now present our definition. Let sender S and receiver R be the participants of the protocol[3]. Let ν be the security parameter and λ be the correctness parameter, upper-bounding error probability by $O(2^{-\lambda})$. Let D_I and D_S be the respective domains of the parties' private inputs and sender's secrets. Let $d_I = |D_I|$ and $d_S = |D_S|$. We assume that both domains are known to both parties. Let R have input $x \in D_I$, and S has input $(y \in D_I, s_0, s_1 \in D_S)$. Let $Q : D_I \times D_I \mapsto \{0, 1\}$ be a predicate. Consider the SCOT functionality:

Functionality 2.

$$f_{Q-\mathrm{SCOT}}(x, (y, s_0, s_1)) = \begin{cases} (s_1, \textit{empty string}) & \textit{if } Q(x, y) = 1, \\ (s_0, \textit{empty string}) & \textit{otherwise} \end{cases} \tag{3.1}$$

[2]Clearly, because secure function evaluation can be based on OT (Kilian [63]), COT implies SCOT. This solution, however, is inefficient.

[3]While our definitions do not impose any round complexity restrictions, our constructions are one-round. Thus, Alice (the party who sends the first message) will be the Receiver, and Bob will be the Sender

There are many models in which we can consider computing this functionality. Each of the two parties may be malicious or semi-honest and each party may or may not be computationally limited[4]. We wish to give one definition that refers to all possible models and rely on existing definitions of secure computations in these models. We refer the reader to Goldreich [49] for in-depth presentations of definitions of security in many interesting models.

Definition 11. *(Q-Strong Conditional Oblivious Transfer)*
We say that a protocol Π *is a* Q-strong conditional oblivious transfer *protocol with respect to a given model, if it securely implements functionality* $f_{Q-\text{SCOT}}$ *(2) in the given model.*

We note that this general definition covers the case when Q is probabilistic.

One of the more practical and interesting settings is the model with the semi-honest unlimited receiver, semi-honest polytime sender and deterministic Q. We discuss our constructions in this model, and thus wish to explicate the definition for this setting.

Definition 12. *Let receiver R, sender S, their inputs x and y, secrets s_1 and s_0, unary parameters ν and λ, and predicate Q be as discussed above. We say that* Π *is a* strong conditional oblivious transfer *protocol for predicate Q in the semi-honest model with computationally unlimited receiver and polytime sender if*

- Transfer Validity. *With overwhelming probability in λ: If $Q(x,y)=1$, R obtains s_1, otherwise R obtains s_0.*

- Security against R. *(R obtains essentially no information other than the transferred secret) There exists a simulator* Sim_R*, such that for any x, y, s, s' from appropriate domains:*

$$\textit{if } Q(x,y) \textit{ then } \{\text{Sim}_R(x,s)\}_\nu \overset{s}{\equiv} \{\text{VIEW}_R^\Pi(x,(y,s',s))\}_\nu$$
$$\textit{if } \neg Q(x,y) \textit{ then } \{\text{Sim}_R(x,s)\}_\nu \overset{s}{\equiv} \{\text{VIEW}_R^\Pi(x,(y,s,s'))\}_\nu$$

[4]Of course, in some of the combinations it is not possible to have nontrivial secure SCOT protocols, such as when both parties are computationally unlimited.

- Security against S. *(S gets no efficiently computable information about x)*

  There exists an efficient simulator Sim_S, such that for any $x, (y, s_0, s_1)$ from appropriate domains:

$$\{\mathrm{Sim}_S(y, s_0, s_1)\}_\nu \overset{c}{\equiv} \{\mathrm{VIEW}_S^\Pi(x, (y, s_0, s_1))\}_\nu.$$

As further justification, we wish to point out an interesting use of Q-SCOT protocols. When sufficiently long secrets are chosen randomly by S, upon completion of a Q-SCOT protocol, R does not know either the value of Q, or the non-transferred secret. Thus this can be viewed as a convenient way to share the value of Q among R and S. Further, the secret that R received may serve as a proof to S of the value of Q. For example, R, by sending the received secret to S, convinces him of the value of Q, even if R may have tried to cheat. This is not possible with COT, as R is only able to provide such proof if $Q(x, y) = 1$.

3.3 The GT-SCOT Protocol

Research specifically addressing the GT problem is quite extensive. It was considered as a special case in the context of general secure function evaluation [5, 70, 76, 85, 90, 89]. This general solution is impractical. However, because the circuit for computing GT is quite small, the general solution based on the natural circuit for computing GT is the best currently known one-round approach in the model with the computationally bounded Alice. As people searched for efficient solutions to special classes of problems in different models, more efficient GT solutions implicitly appeared. Naor and Nissim [73] presented a general approach to securely computing functions with low communication overhead. While the application of their solution to GT is quite efficient in the message length, it needs at least $O(\log n + \log \frac{1}{\epsilon})$ 1-out-of-$O(n)$ oblivious transfers and the same number of rounds, where ϵ is the tolerated probability of error.

Fischlin [41] proposed a solution that significantly reduced the number of modular multiplications, while also reducing the message size and maintaining the minimal one-round efficiency. This is the best previously known solution to the GT problem in the model with unbounded Alice. The number of modular multiplications required to complete his protocol is $6n\lambda + n\lambda \log N$, where $2^{-\lambda}$ is the allowed error probability. The message complexity (in bits) is $n \log N(\lambda + 1)$. Fischlin also extends this protocol (at the cost of approximately doubling the communication and computation costs) to satisfy our definition of GT-SCOT, with the exception of leaking the value of the predicate. We remark that this extension can be further extended to fully satisfy our definitions at the expense of further approximately doubling the communication and computation costs.

Di Crescenzo [33] proposed a GT protocol, secure according to our SCOT definition. Its cost is $12n^2 + 8n^2 \log N$ modular multiplications and message complexity is $8n^2 \log N$ bits.

3.3.1 Our Construction

Our constructions use semantically secure additively homomorphic encryption schemes with large message domains. For the ease and clarity of presentation and to enable resource analysis, we "instantiate" our protocols with the original Paillier scheme. We remark that the Paillier scheme has received much attention in the literature recently, and several variants, including an elliptic curve version [44], have appeared. Using more efficient implementations may further improve our results.

Let $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ be the instance generation, encryption and decryption algorithms, respectively, of such a scheme. As in Definition 12, let R and S be the receiver and the sender with inputs x and y respectively and common parameters ν and λ. Let $x, y \in D_I$ and $s_0, s_1 \in D_S$. Let $d_S = |D_S|$ and, without loss of generality, $d_I = |D_I| = 2^n$.

Throughout this section, we will work with numbers which we will need to represent as binary vectors. For $x \in \mathbb{N}$, unless specified otherwise, x_i will denote the i^{th} most significant bit in the n-bit binary representation of x, including leading zeros, if applicable. Where it is clear from the context, by x we mean the vector $\langle x_1, x_2, ..., x_n \rangle$ of bits, and by $\text{Enc}(x)$ we mean a vector $\langle \text{Enc}(x_1), \text{Enc}(x_2), ..., \text{Enc}(x_n) \rangle$. We will also write $\text{Enc}(x)$ instead of $\text{Enc}_{pk}(x)$, where pk is clear from the context.

For the clarity of presentation, we describe the setup phase outside of the protocol. We stress that it is run as part of R's first move, and in particular, *after* the parties' inputs x, and (y, s_0, s_1) have been fixed.

Setup Phase. R sets up the Paillier encryption scheme with group size $N = pq$ by running Gen and generating secret and public keys (sk and pk). He chooses the number of bits in N to be $\max\{\nu, |d_S| + \lambda\}$, where $|d_S|$ is the bit-length of the integer d_S.

We will view D_S as a subset of $\mathbb{Z}_N$, and will perform operations on elements of D_S modulo N.

Observation 1. *We envision the following practical parameter choices for our GT protocols. First, choose N and λ to satisfy the security and correctness requirements of the encryption scheme. In practice, $\log N (\approx 1000) \gg \lambda (\approx 40..80)$, so we set $|d_S| = \log N - \lambda > 900$ bits of the bandwidth of the encryption scheme to be used for sending secrets. If D_S needs to be much larger than that, it may be more practical to split it in blocks of size $|d_S|$ and run GT-SCOT several times. Choosing parameters in this manner also simplifies comparison of our results to others, and we follow this approach in Sect. 3.4.2.*

Observation 2. *There is a negligible (in λ) minority of elements of D_S in the group of size N.*

For our protocols, we are only interested in binary comparisons, i.e. one of $\{>, <, \leq, \geq\}$. We can trivially reduce $\{\geq, \leq\}$ to $\{>, <\}$. Furthermore, we assume that $x \neq y$. This can be enforced by mapping, for instance, $x \mapsto 2x, y \mapsto 2y+1$. Similarly, we assume that $s_0 \neq s_1$. The case when $s_0 = s_1$ can be reduced to the $s_0 \neq s_1$ case by, for example, S setting $y = \max\{D_I\}$ and $s_1 \in_R D_S \setminus \{s_0\}$, ensuring that $x < y$ and s_0 is always sent.

We now present the GT-SCOT construction. The intuition behind each step is presented immediately below, in the proof of the corresponding security theorem. Note that the arithmetic operations in the presentation of the construction are in the group of the plaintext domain of the encryption scheme.

Construction 1. *(Computing functionality GT-SCOT)*

1. *R runs the setup phase, then encrypts each bit x_i of x with the generated pk and sends $(pk, Enc(x_1), ..., Enc(x_n))$ to S.*

2. *S computes the following, for each $i = 1..n$:*

   (a) an encryption of the difference vector d, where $d_i = x_i - y_i$.

   (b) an encryption of the flag vector f, where $f_i = x_i \, XOR \, y_i = (x_i - y_i)^2 = x_i - 2x_iy_i + y_i$.

   (c) an encryption of vector γ, where $\gamma_0 = 0$ and $\gamma_i = 2\gamma_{i-1} + f_i$.

   (d) an encryption of vector δ, where $\delta_i = d_i + r_i(\gamma_i - 1)$, where $r_i \in_R \mathbb{Z}_N$.

   (e) a random encryption of vector μ, where $\mu_i = \frac{s_1 - s_0}{2}\delta_i + \frac{s_1+s_0}{2}$

   and sends a random permutation $\pi(Enc(\mu))$ to R.

3. *R obtains $\pi(Enc(\mu))$, decrypts it, and determines the output as follows: if μ contains a single $v \in D_S$, output v, otherwise abort.*

Theorem 1. *The protocol of Construction 1 is a GT-SCOT protocol in the semi-honest model, assuming semantic security of the employed encryption scheme.*

Proof. We will first show that the protocol correctly computes the desired functionality. This part of the proof also provides the intuition of the construction.

It is easy to see that the homomorphic properties of the encryption scheme allow S to perform all necessary operations. For example, step 2b is possible because y_i are known to S.

Step 2a computes the (encryption of) the bit difference vector d.

Observe that the flag vector f, whose encryption is computed in Step 2b, is a $\{0,1\}$-vector, with the ones in positions where x and y differ. Furthermore, γ, whose encryption is computed in Step 2c, is a vector with the following structure: it starts with zero or more zeros, then a one, then a sequence of non-ones. Moreover, with overwhelming probability the non-zero elements $(\gamma_i - 1)$ are not multiples of either p or q, i.e. are in $\mathbb{Z}_N^*$. This is because the fraction of multiples of p or q in $\mathbb{Z}_N$ is negligible, and p and q are chosen randomly and independently of x and y.

Let ind_1 be the (only) position where $\gamma_{\mathrm{ind}_1} = 1$. This position is where x and y first differ, and thus d_{ind_1} determines $\mathrm{GT}(x,y)$. The transformation $(\gamma, d) \to \delta$ of Step 2d randomizes all coordinates of δ, while setting δ_{ind_1} to the value of d_{ind_1}. Because, with overwhelming probability, $(\gamma_i - 1) \in \mathbb{Z}_N^*$, multiplying it by $r_i \in_R \mathbb{Z}_N$ randomizes δ perfectly in $\mathbb{Z}_N$.

With overwhelming probability, the transformation $(\delta, s_0, s_1) \to \mu$ of Step 2e is a permutation on $\mathbb{Z}_N$ that maps $-1 \mapsto s_0, 1 \mapsto s_1$. Indeed, it is not such a permutation only when $(s_1 - s_0)$ is a multiple of p or q, an event that occurs with negligible probability, because p and q are are chosen randomly and independently of s_1 and s_0. This permutation preserves the randomness properties of all elements of the vector, and (as is easy to verify) performs the mapping we are looking for. Random re-encryption of Step 2e hides the information that may be contained in the randomness of the encryption. Finally, the random permutation $\pi(\mu)$ of Step 2 hides the index of the determining d_i.

Consider the probability that there is not exactly one element of size $|d_S|$ in the vector decrypted by R. It easily follows from Observation 2 that this probability is negligible. Thus, with overwhelming probability, R terminates and outputs the correct value.

Security of R (against the semi-honest S) trivially holds because of the semantic security properties of the employed encryption scheme.

We now prove security of S against an unlimited semi-honest R by constructing a protocol view simulator $\mathrm{Sim}_R(x, s)$, where x is the input, and s is the output of the protocol. $\mathrm{Sim}_R(x, s)$ has to generate a distribution statistically close to the view of R in a real execution - $\mathrm{VIEW}_R(x, (y, s_0, s_1)) = \{x, r, \mathrm{Enc}(\pi(\mu))\}$, where r is the randomness used by R to generate pk and sk (of the setup phase) and the random encryptions of the first message, and $\pi(\mu)$ is defined in the protocol construction. $\mathrm{Sim}_R(x, s)$ proceeds as follows. It first generates a random string r' of appropriate length (to match r). It uses r' to compute the keys sk and pk (including N). It then computes a candidate μ': for $i = 1..n$, pick random $\mu'_i \in_R \mathbb{Z}_N$. It then replaces a random element of μ' with the received s, and outputs $\{x, r', \mathrm{Enc}_{pk'}(\mu')\}$, where $\mathrm{Enc}_{pk'}(\mu')$ is a vector of random encryptions of coordinates of μ' under the pk'. Because of the previously presented arguments of the randomness of all elements of $\pi(\mu)$ (other than the one that carries the secret) and the randomness of re-encryption, it is easy to see that Sim_R generates a distribution statistically close to the view of R. We note that the simulation is not perfect, since the transfer of the other secret is possible during the real execution, with negligible probability. $\square$

We observe that a GT-SCOT protocol, such as presented above, immediately implies solution to GT, in the semi-honest model. Indeed, running GT-SCOT with at least one of the secrets s_i known to R (say $s_1 = 1$), immediately yields the desired functionality. Moreover, for GT, the transformation of step 2e is unnecessary (while the re-randomization of the same step is still required).

3.3.2 Resource Analysis

We evaluate the message and modular multiplication efficiency of our construction based on the use of Paillier encryption scheme. We note that we do not include the relatively small computational cost of key generation, to be consistent with the compared results of [32], [33] and [41]. Let n be the length of inputs x and y in binary, N-the size of the plaintext domain of the Paillier scheme. Then message complexity of Construction 1 is $l = 2n \log(N^2) = 4n \log N$ bits.

Let $w = w(y) \leq n$ be the weight (i.e. the number of ones) of the binary representation of y. To encrypt each bit, $\log N$ multiplications are required. Observe that it is not necessary to perform expensive randomized encryption in the intermediate steps of S. This allows us to make do with only w multiplications for each of the steps 2a, 2b, $2n$ - for step 2c, and $(\log N + 2)n$ - for step 2d, and $(|s_i| + \log N)n \leq 2n \log N$ - for step 2e of the protocol. We note that if we do not perform the transformation of step 2e (when, for example, computing GT), we only need $n \log N$ multiplications for the last step.

Decryption takes $2n \log N$ multiplications. Thus, in total, the protocol requires no more than $(5n + 1) \log N + 6n$ modular multiplications ($(4n + 1) \log N + 6n$ for GT). We stress that transferring up to $\log N - \lambda$ bit secrets requires the same resources. We observe that the encryption and re-encryption multiplications can be precomputed once the encryption scheme is initialized.

We compare the efficiency of our approach to that of Fischlin [41] and Di Crescenzo [33], using appropriate parameters. We first note that in practice, no known attack on the Paillier system is better than factoring the modulus N. Clearly, factoring based attacks would also be effective against the GM scheme with the same modulus size. Thus, having already assumed CCRA (see Sect. 2.7.2), we also assume that the security of Paillier and GM schemes with the modulus of the same size are approximately the same. We note that our modular multiplications are four times slower, since we are working with modulus length twice that of the Goldwasser-Micali encryption scheme employed in [41] and [33]. The comparisons are summarized in the Table in Sect. 3.4.2.

3.4 SCOT for Unions of Intervals

In this section we present new efficient protocols for I-SCOT (SCOT based on the membership in an interval) and UI-SCOT (SCOT based on the membership in a union of intervals), both of which are generalizations of GT-SCOT. More specifically, in I-SCOT, the secret s_1 (resp. s_0) is transferred to R if R's input point x belongs (resp. doesn't belong) to the interval that is the input of S. Similarly, in UI-SCOT, the secrets are transferred based on whether R's input point x belongs to the set of intervals that is the input of S. We will sometimes write k-UI-SCOT, to emphasize that S's input to UI-SCOT has k intervals.

We build the I-SCOT and UI-SCOT protocols on our GT-SCOT solution. While other GT-SCOT approaches (such as based on Fischlin's protocol) are also suitable for these constructions, our solution is simpler and produces more efficient protocols in terms of both multiplication and communication complexity. In our constructions, we denote the instance of the Q-SCOT functionality with the secrets s_0, s_1 on parties' inputs x, y by Q-SCOT $(s_1|s_0?Q(x,y))$.

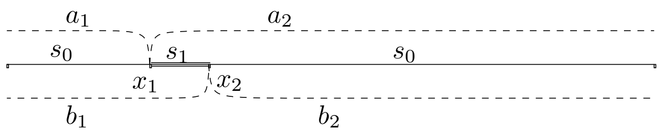
In Sect. 3.4.1 we show how to reduce UI-SCOT to I-SCOT and I-SCOT to GT-SCOT. (We chose this modular way of presentation because it appears to be simpler.) We note that in our model, secure reductions provide us with secure protocols when the underlying oracles are replaced by their secure implementations (see Goldreich [49] for the composition theorem.) Furthermore, in our model the oracles' implementations may be run in parallel, which, with our implementations, provides secure one-round protocols for I-SCOT to UI-SCOT.

3.4.1 The UI-SCOT protocol

In the I-SCOT setting, S's input $x_1, x_2 \in D_I$ represents an interval I. We need S to obliviously transfer to R the secret s_1 (resp. s_0) if $x \in I$ (resp. $x \notin I$), for R's input

$x \in D_I$. Without loss of generality, we assume that the domain of secrets D_S is an additive group[5] $\mathbb{Z}^+_{d_S}$. In our discussion, all additions of secrets will be done in D_S, unless specified otherwise.

The intuition of the reduction of I-SCOT to GT-SCOT is as follows, illustrated on the diagram below. Interval I splits D_I in three parts, and S wishes to transfer s_1 "on the central part" (I) and s_0 "on the side parts" ($D_I \setminus I$). The idea is to represent these secrets as sums of independently random (i.e. random if taken separately) elements ($a_1, a_2, b_1, b_2 \in D_S$) which are to be transferred using GT-SCOT.



Construction 2. *(Reducing I-SCOT to GT-SCOT)*

1. *S randomly chooses $a_1 \in D_S$ and sets $b_1, a_2, b_2 \in D_S$ to satisfy $s_0 = a_1 + b_1 = a_2 + b_2$ and $s_1 = a_2 + b_1$*

2. *-Reduction: R and S (in parallel) invoke oracles for GT-SCOT($a_1|a_2?x < x_1$) and GT-SCOT($b_1|b_2?x < x_2$).*

3. *R obtains $a', b' \in D_S$ from GT-SCOT oracle executions and outputs $a' + b'$.*

Theorem 2. *The protocol of Construction 2 securely reduces functionality I-SCOT to GT-SCOT in the semi-honest model.*

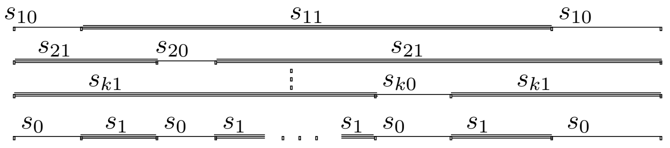
Proof. The transfer validity property of this reduction trivially holds. Since S does not receive any messages from R or oracle executions, the reduction is secure against semi-honest S. We show how to construct Sim_R, simulating the following ensemble (view of R): $\mathrm{VIEW}_R(x, (x_1, x_2, s_0, s_1)) = \{x, r_1, r_2\}$, where r_1, r_2 are the sent (via the GT-SCOT oracles) a_i, b_j. Let s be the transferred secret. Then $\mathrm{Sim}_R(x, s) = \{x, r'_1, r'_2\}$, where r'_i

[5]We stress that we use GT-SCOT as black box, and, in particular, addition in D_S is unrelated to the corresponding operation in the GT-SCOT implementation.

are independently random elements of D_S that sum up to s. Because, by construction, r_1, r_2 are also independently random with the same sum, Sim_R perfectly simulates view of R. □

We now wish to reduce UI-SCOT of polynomially many intervals to I-SCOT. Here, S's input represents a set of *disjoint* intervals $\{I_i = (x_{i1}, x_{i2} \in D_I)\}$, and the secrets $s_0, s_1 \in D_S$. S wishes to transfer s_1 if $x \in \bigcup I_i$, and transfer s_0 otherwise. Let k be the number of intervals in the set (to avoid leaking k to R, S can pad it to a known upper bound by adding empty intervals).

We represent $\bigcup I_i$ as the intersection of k intervals as follows, which is also illustrated on the diagram below. The bottom line represents the input set of intervals on the domain, and all other lines represent the constructed intervals that together correspond to this set. (All intervals are marked by bold lines.) The s_i are the secrets to be transferred by the UI-SCOT construction, and the s_{ij} are the intermediate secrets to be created by UI-SCOT and transferred by the existing I-SCOT protocol. Because the input intervals are disjoint, the cut out (thin, on the diagram) parts of the constructed intervals do not intersect, and thus any x either belongs to all or to all but one constructed intervals.



To reduce UI-SCOT to I-SCOT, we need to choose $s_{ij} \in D_S$ based on the given s_i. Because of the above observation we only need to satisfy the following: $s_1 = \sum_i s_{i1}$ and $s_0 = (\sum_{i \neq j} s_{i1}) + s_{j0}, \forall j = 1..k$ Observe that the second condition is equivalent to requiring $s_1 - s_0 = s_{j1} - s_{j0}, \forall j = 1..k$.

Construction 3. *(Reducing UI-SCOT to I-SCOT)*

1. *S chooses $s_{11}, ..., s_{(k-1)1} \in_R D_S$ and sets $s_{k1} = s_1 - \sum_{i=1..k-1} s_{i1}$ and $s_{i0} = s_{i1} - (s_1 - s_0), i = 1..k$.*

2. -Reduction: S and R (in parallel) invoke oracles for I-SCOT$(s_{i1}|s_{i0}?x \in I_i)$, for each $i = 1..k$.

3. R obtains $a_1, ..., a_k \in D_S$ from k oracle executions and outputs $\sum_i a_i$.

Theorem 3. *The protocol of Construction 3 securely reduces functionality UI-SCOT to I-SCOT in the semi-honest model.*

Proof. The transfer validity property of this reduction trivially holds. Since S does not receive any messages from R or oracle executions, the reduction is secure against semi-honest S. We show how to construct Sim_R simulating the view of R $\text{VIEW}_R(x, y) = \{x, r_1, ..., r_k\}$, where $r_1, ..., r_k$ are the oracle sent elements of D_S defined by step 1 of the construction. Let s be the transferred secret. Then $\text{Sim}_R(x, s) = \{x, r'_1, ..., r'_k\}$, where $r'_i \in_R D_S$ with the restriction $s = \sum_i r_i$. Sim_R perfectly simulates view of R because both ensembles are $(k-1)$-wise independent random numbers that sum up to the same value s. $\square$

The $(\bigwedge_i Q_i(x_i, y_i))$-COT Protocol.

We now build $\bigwedge_i Q_i(x_i, y_i))$-COT using oracles for corresponding Q_i-SCOT. R now has input $x_1, ..., x_n$, and S has $y_1, ..., y_n$. S wishes to send a secret s to R iff $\bigwedge_i(Q_i(x_i, y_i)) = 1$. The idea is to introduce "specialness" of s like we did for GT-SCOT, by, for example, extending the domain of secrets D_S to group $D'_S = \mathbb{Z}^+_{d'_S}$, where $d'_S = |D'_S| \gg |D_S|$, Then S represents $s \in D_S$ as a sum of random secrets $s_i \in_R D'_S$, and runs Q_i-SCOT$(s_i|r_i?Q_i(x_i, y_i))$, where $r_i \in_R D'_S$. Indeed, if the conjunction holds, then only the s_i's will be transferred, and they will sum up to $s \in D_S$. If any (or any number of) predicates do not hold, one (or more) r_i will be transferred, which will randomize (in D'_S) the sum obtained by R.

Construction 4. *(Reducing $(\bigwedge_i Q_i(x_i, y_i))$-COT to Q_i-SCOT)*

1. S chooses $r_1, ..., r_n, s_1, ..., s_{n-1} \in_R D'_S$ and sets (in D'_S) $s_n = s - \sum_{i=1..n-1} s_i$.

2. *R and S in parallel invoke oracles for Q_i-SCOT$(s_i|r_i?Q_i(x_i, y_i))$, $\forall i = 1..n$.*

3. *R obtains $a_1, ..., a_n \in D'_S$ from the Q_i-SCOT oracle executions and sets $v = \sum_i a_i$. R outputs v, if $v \in D_S$, and outputs $\perp$ otherwise.*

Theorem 4. *The protocol of Construction 4 securely reduces functionality $(\bigwedge_i Q_i(x_i, y_i))$-COT to Q_i-SCOT in the semi-honest model.*

Proof: The simple proof is very similar to the previous ones and is omitted. □

Corollary 1. *There exists (via construction 4 and DeMorgan laws) efficient one-round protocols for computing conjunction and disjunction of memberships in sets of intervals, secure against computationally unlimited R.*

3.4.2 Resource Analysis

We continue and expand the resource analysis of Sect. 3.3.2. Recall that λ and ν are the correctness and security parameters, n is the number of bits in the compared numbers, and N is the modulus of the employed encryption scheme. As discussed in Observation 1, we choose $\nu = \log N$ and λ as in [41]. This determines the secrets domain D_S to be of size $2^{\nu-\lambda}$. As noted in Sect. 3.3.2, we do not include the cost of key generation in any of the compared solutions.

It is easy to see that our k-UI-SCOT construction (Constr. 3) makes $2k$ calls to the underlying λ-bit GT-COT oracle. Thus, when using our implementation of GT-SCOT, UI-SCOT requires sending $8kn \log N$ bits and performing about $40kn \log N$ multiplications in group of size N. Using λ-bit GT-SCOT oracle implementation based on Fischlin's and Di Crescenzo's GT results in almost full factor of $2k$ blowup in communication since server sends most of the traffic. The $2k$ factor blowup in the computation also seems necessary when using these schemes.

The following table summarizes the cost of comparable modular multiplications and communication of our protocol in relation to others.

Protocol	GT predicate		c-bit GT-SCOT, $c<\nu$-λ	
	mod. mult.	comm.	mod. mult.	comm.
of [41]	$6n\lambda + n\lambda \log N$	$\lambda n \log N$	$24nc\lambda + 4nc\lambda \log N$	$4nc\lambda \log N$
of [32]	$8n + 4n \log N$	$4n \log N$	N/A	N/A
of [33]	$6n^2 + 4n^2 \log N$	$4n^2 \log N$	$12n^2c + 8n^2c \log N$	$8n^2c \log N$
our work	$16n \log N$	$4n \log N$	$20n \log N$	$4n \log N$

Protocol	k-UI-SCOT	
	mod. mult.	comm.
of [41]	$48kn\lambda^2 + 8kn\lambda^2 \log N$	$8kn\lambda^2 \log N$
of [32]	N/A	N/A
of [33]	$24kn^2c + 16kn^2c \log N$	$16kn^2c \log N$
our work	$40kn \log N$	$8kn \log N$

We see no obvious way to transform the schemes of [32] to GT-SCOT, and thus do not include the corresponding resource calculations.

Chapter 4

Comparing Encrypted Numbers

4.1 Introduction

Summary of the contributions of the chapter. We consider the problem of comparing two encrypted numbers and its extension – transferring one of the two secrets, depending on the result of comparison. We show how to efficiently apply our solutions to practical settings, such as auctions with the semi-honest auctioneer, proxy selling, etc. We propose a new primitive, *Conditional Encrypted Mapping*, which captures common security properties of one round protocols in a variety of settings, which may be of independent interest.

4.1.1 Motivation of the Problem and the Setting

In this chapter we continue studying secure evaluation of the Greater Than (GT) predicate. Recall, it is one of the most basic and widely used functionalities. It plays an especially important role in secure financial transactions and database mining applications.

Auctions and Bargaining. With the continuing expansion of the Internet, electronic commerce and especially online auctions continue to grow at an impressive pace.

Many sellers also discover the appeal of flexible pricing. For example, sites such as priceline.com ask a buyer for a price he is willing to pay for a product, and the deal is committed to if that price is greater than a certain (secret) threshold.

In many such situations, it is vital to maintain the privacy of bids of the players. Indeed, revealing an item's worth can result in artificially high prices or low bids, specifically targeted for a particular buyer or seller. While a winning bid or a committed deal may necessarily reveal the cost of the transaction, it is highly desirable to keep all other information (e.g. unsuccessful bids) secret.

There has been a large stream of work dedicated to ensuring privacy and security of online auctions and haggling (e.g.,[21, 33, 41, 76]). Our work complements, extends, and builds on it. We discuss the Private Selective Payments protocols of Di Crescenzo [33] and show how our improvements benefit this application.

The need for comparing encrypted numbers. It is often beneficial to both sellers and buyers to employ a mutually semi-trusted server S to assist them in their transaction. The use of such a server simplifies secure protocol design, allowing for more efficient protocols. It allows the seller to be offline most of the time, allowing S to act on behalf of the seller in handling bid requests. Further, a reputable S (such as eBay) may provide additional assurance of security to the potential buyer. However, since sellers and buyers wish to hide their inputs from S, the latter must work with, e.g. compare, *encrypted* numbers. We propose a formalization of this setting, as well as new, more efficient GT protocols for it. We note that the concept of server-aided computation is not new, and is subject of considerable amount of research (e.g. [39, 71]).

Other applications. In Sect. 3.1.1, we mentioned other interesting applications (distributed database mining, set intersection, etc.) that benefit from efficient secure evaluation of GT. These applications might need to employ a proxy server S, as above; if so, the work of this chapter improves their performance as well.

4.1.2 Our Contributions, Setting and Outline of the Work

We approach several practical problems (auctions, proxy selling, GT) in a variety of settings, concentrating on a setting with a semi-honest helping server.

We are interested in one-round protocols, where clients send their encrypted inputs to a "cypto computer" S, who produces an output that can be decoded by the clients. Such scenarios arise in a variety of practical settings. To enable formal discussion of crucial parts of our protocols in a number of settings simultaneously, we extract what these settings have in common – the following requirements on the output of S: it allows the reconstruction of the value of the function, and does not contain any other information. This allows us to postpone the (easy but tedious) discussion of setting-specific clients' privacy requirements. We formalize (Def. 13) a special case of this notion, which we call *Conditional Encrypted Mapping* (CEM). Here, S has two secrets s_0, s_1, is given encryptions of two values x, y, and outputs something that allows (only) reconstruction of $s_{Q(x,y)}$, where Q is a fixed public predicate. We note that our statistical privacy requirement on the output of S is very strong, e.g., precluding Yao's garbled circuit-based solutions.

We propose two new, more efficient CEM protocols for the GT predicate (Sect. 4.4). We use ideas of our protocol of Chapter 3. Recall, that protocol requires S to know one of the compared numbers, and thus cannot be naturally cast as a CEM. We overcome this with a new tool – a randomized way to represent secrets to be transferred by S (presented in Sect. 4.4.3). The cost of the new solution is comparable to that of Chapter 3. We believe this method may be used to improve efficiency of other constructions relying on homomorphic encryptions.

In Sect. 4.5, we show how our constructions result in new, more efficient, protocols for the examples of private selective payments of Di Crescenzo [33] and proxy selling. We discuss methods of protection against malicious behavior of parties. We mention that efficient CEM schemes exist for any NC^1 predicate (Sect. 4.4.7).

In Sect. 4.6 we summarize and compare resource requirements of schemes based on the work of Di Crescenzo [33], Fischlin [41], Laur and Lipmaa [68] and ours.

4.2 Related Work

We discuss related work in both directions of our contributions – definition of CEM and concrete protocols for auction-like functionalities.

Variants of CEM. Several notions similar to CEM were previously proposed.

The notion of Conditional Oblivious Transfer (COT) was introduced by Di Crescenzo, Ostrovsky and Rajagopalan [32] in the context of timed-release encryption. It is a variant of Oblivious Transfer (OT) [81]. Intuitively, in COT, the two participants, a receiver R and a sender S, have private inputs x and y respectively, and share a public predicate $Q(\cdot,\cdot)$. S has a secret s he wishes (obliviously to himself) to transfer to R iff $Q(x,y)=1$. If $Q(x,y)=0$, no information about s is transferred to R. R's private input and the value of the predicate remain computationally hidden from S.

A similar notion to COT, Conditional Disclosure of Secrets (CDS), was introduced by Gertner, Ishai, Kushilevitz and Malkin [46] in the context of multi-server Symmetrically Private Information Retrieval (SPIR). In their work, the receiver of the secret apriori knows the inputs of the (many) senders. The secret is unknown to the receiver and sent to him only if a predicate holds on the inputs.

Aiello, Ishai and Reingold [1] adapt CDS into the single server setting, where the (single) sender holds *encryptions* of parts (i.e. bits) of input. The receiver knows both the input and the decryption key. Again, the receiver does not know the secret; it is sent to him only if a predicate holds on the input.

Laur and Lipmaa [68] extend the study of CDS for the case of additive homomorphic encryptions, give generic constructions and specific protocols (GT).

The lack of requirement of privacy of the value of $Q(x,y)$ and the sender's input often prevents the use of COT or CDS as a building block of other protocols. Di Crescenzo [33] described a stronger concept, Symmetrically-private COT, by additionally requiring that both parties' inputs x, y remain private. Later, we independently proposed and formalized a similar notion, which we call Strong COT (see Chapter 3 or [14]). Of the above, CEM is most similar to this notion. We note that CEM is a stronger (than SCOT) notion, explicitly allowing reuse of generated encryption keys in multiple executions. We also have the feature of not specifying the precise security properties of the used encryptions, allowing for more flexibility and applicability (see Sect. 4.1.2 and 4.3 for more discussion).

Auctions and Private Selective Payments Protocols (PSPP). PSPP, introduced by Di Crescenzo [33], solve the following practical problem. A server has a private message representing, say, a signed authorization, and wants to give it to one among several clients, according to some public criteria, evaluated on the server's and clients' private inputs. Client's inputs may represent their auction bids, and a server's input may be a lowest acceptable price or a required signature. Di Crescenzo considers a natural instance of PSPP, where the highest bidding client obtains the authorization. He considers a setting with a helping semi-honest server and malicious clients.

Di Crescenzo designs his protocols in several phases. During *registration*, executed between each client and the server, the client's public/private key pair is established, and the server obtains the public key. Then the *selection* protocol is executed between all registered clients and the server, during which the selected client obtains the server's secret. Finally, in the *verification* phase, the selected client presents his claim – the obtained secret – and convinces the server that he indeed is the selected client. The registration and verification phases are designed using standard cryptographic tools; it is the selection phase that is the challenging computationally expensive area. The main contribution of [33] is the novel maximum bidder selection protocols.

Our main contribution, GT-CEM constructions, can be used to replace the core –

the selection protocols – of the PSPP of [33] (with corresponding natural modifications of the other two phases). Appropriately modified protocols of Fischlin [41] and Laur and Lipmaa [68] can be similarly used. We discuss more details and the resulting efficiency improvements of our protocols in Sect. 4.5 and 4.6.

We also mention the following related work in settings significantly different from ours. Naor, Pinkas and Sumner [76] use Yao's garbled circuit approach in the setting with a semi-honest mostly offline server, whose role is to ensure that the auctioneer does not cheat. Cachin [21] suggested a protocol for private bidding with the semi-honest server in the setting where the bidders additionally exchange messages among each other.

4.3 Conditional Encrypted Mapping

We consider the setting where one of the players is a facilitator of the computation of the multi-party functionality f. This player – the Server S – is given the encrypted inputs to f; he produces some representation of the value of f. The value of f can later be decoded from this representation using the private key of the employed encryption scheme. This scenario is appealing for its round efficiency and is widely applicable in practice. For example, it applies to auctions with semi-honest servers. There, the server S is given encryptions of parties' bids, and he wants to commit to a deal (e.g. by sending a secret) with the winner.

The first step in designing secure protocols is making explicit the setting in which they are run and the necessary security requirements. This is a difficult task, especially since we would like our constructions to be applicable to a variety of settings. For example, the server S may obtain encrypted inputs from parties A and B and let either A or B or a third party C decode the output. Protocols can use encryption schemes, which may or may not be re-initialized for each execution of the protocol. Players A, B or C may have different levels of trust.

Encompassing all these situations in one definition is difficult. We propose to extract and formalize what these definitions would have in common – requirements of correctness and privacy of the output of the semi-honest Server S. This modularity is very convenient, since we can now model S as a non-interactive algorithm. A variety of setting-specific requirements for hiding the input from the server can be later defined and satisfied with appropriate use of encryption.

Encrypted Mapping. We model the Server S as a polytime randomized mapping algorithm *Rmap*. *Rmap* takes as input the public key of the encryption scheme E, the (encrypted with E) input(s), and outputs some representation of the value of f. Of course, this output should be interpreted. We require existence of the polytime recovery procedure *Rec*, which takes the representation of the value of f and the private key of E and computes the intended output (this is the correctness condition). Further, we require that the randomized representation statistically hides all other information, ensuring privacy of arbitrary compositions of outputs of *Rmap* even against computationally unlimited attackers. We call the pair $(\mathit{Rmap}, \mathit{Rec})$ an Encrypted Mapping (EM). We formalize a variant of this notion in Def. 13 below.

We consider it advantageous not to specify the requirements of security of encryption in the definition, for the following reason. It allows a protocol designer to concentrate on the high-level combinatorial properties of EM and defer discussion of detailed setting-specific concerns. Such low-level concerns include considering whether some inputs to S contain decryption keys (which would allow S to learn more than he should) and considering malicious behaviour, such as providing invalid or substituted inputs. A protocol designer can now first describe the combinatorial *Rmap* and *Rec*, which would imply solutions to a variety of settings in the semi-honest model, assuming the semantic security of the employed encryption scheme. Afterwards, the protocols can be adapted to a variety of specific settings and modified to withstand certain malicious behaviours (e.g. using the conditional disclosure techniques of [1, 68]. See more in Sect. 4.5.1).

We wish to give a very strong definition, so that the constructions can be used in a variety of settings. In particular, we want our construction to work with all instantiations of used encryption schemes. In many popular encryption schemes (e.g. Paillier [78]) the plaintext domain D_P varies with different instantiations. Many interesting functions f are defined on fixed domains, independent of D_P. We handle this detail by ensuring that D_P includes the domain of inputs to f by appropriately modifying the family of encryptions to only include members with sufficiently large D_P. We note that a sufficiently large D_P is usually implied by the semantic security requirement of the scheme.

We remark that we achieve a very strong definition by quantifying over all valid inputs and randomness used by encryptions – i.e. over everything but the randomness used by *Rmap*. This, for example, ensures that adversary does not benefit from knowing (and even choosing) the randomness used for encrypting inputs to *Rmap*.

Conditional Encrypted Mapping. In this work, we are mainly interested in constructing the protocols for transferring a secret (e.g. a sale commitment or a rejection) depending on whether a certain predicate on two inputs (e.g. the bid is greater than the asking price) holds. We call the corresponding EM a Conditional Encrypted Mapping (CEM). We give a formal definition for this special case and note that a more general EM definition can be naturally constructed.

We define CEM with respect to an encryption scheme $E = (Gen, Enc, Dec)$. Denote by (sk, pk) a public/private key pair for E, and by E_{pk} denote the initialized encryption scheme E. Let $D_{P_{pk}}$ denote the plaintext domain of E_{pk} and $D_{R_{pk}}$ denote the domain of randomness used by Enc_{pk}. Denote by $Enc_{pk,\alpha}(x)$ the encryption of x under pk using randomness α. Let $Q : D_Q \times D_Q \mapsto \{0, 1\}$ be a deterministic predicate defined on a fixed domain. Recall, we only consider families of E_{pk} where $D_Q \subset D_{P_{pk}}$. Let ν be the security parameter[1]. Let D_S be the (fixed) domain of secrets[2].

Definition 13. *(Q-Conditional Encrypted Mapping) A Q* - Conditional Encrypted Mapping *(Q-CEM) is a pair of polytime algorithms* $(Rmap, Rec)$ *(with implicitly defined domain of mappings $D_{M_{pk}}$), such that the following holds.*

The probabilistic randomized mapping algorithm Rmap takes as input (s_0, s_1, e_0, e_1, pk), where e_0, e_1 are encryptions under E_{pk}, and $s_0, s_1 \in D_S$. Rmap outputs an element from $D_{M_{pk}}$. The deterministic recovery algorithm Rec takes as input secret key sk and an element from $D_{M_{pk}}$ and outputs an element from the domain of secrets D_S or a failure symbol $\perp$.

Rmap and Rec satisfy the following conditions:

- *(correctness)* $\forall (sk, pk) \leftarrow Gen(\nu), \forall s_0, s_1 \in D_S, \forall \alpha, \beta \in D_{R_{pk}}, \forall x, y \in D_Q$: *with overwhelming probability in ν, taken over random inputs of Rmap:*
  $Rec(Rmap(s_0, s_1, Enc_{pk,\alpha}(x), Enc_{pk,\beta}(y), pk), sk) = s_{Q(x,y)}$.

- *(statistical privacy)* $\exists$ *Sim, s.t.* $\forall (sk, pk) \leftarrow Gen(\nu), \forall s_0, s_1 \in D_S, \forall x, y \in D_Q$, $\forall \alpha, \beta \in D_{R_{pk}}$: *the statistical distance*
  $Dist(Sim(s_{Q(x,y)}, pk), Rmap(s_0, s_1, Enc_{pk,\alpha}(x), Enc_{pk,\beta}(y), pk))$
  is negligible in ν.

Note, Def. 13 does not require E to have any security properties. Thus, formally, inputs e_0, e_1 to *Rmap* are simply *encodings* of elements in D_Q (and Q-CEM can be constructed unconditionally). In practice, however, we envision using a semantically secure E; thus we call e_0, e_1 encryptions. Jumping ahead, we note that in our GT constructions of Sect. 4.4.4, the inputs e_0, e_1 to *Rmap* are bitwise encryptions of the

[1]In practice, we are also interested in the correctness parameter λ. Security and correctness properties of Def. 13 are formulated with the notion of statistical closeness. Since ν and λ are polynomially related, we, for simplicity, use only the parameter ν.

[2]Even though for simplicity of presentation the domains D_Q and D_S are fixed, their elements representation is polynomially related to all other parameters. Further, in practice (and in our constructions), D_Q and D_S can grow with ν at no extra cost.

clients' bids. Note that Def. 13 allows this interpretation, since encrypting x bit-by-bit can be viewed as an encryption scheme itself.

Further, Def. 13 does not guarantee either correctness or privacy if e_0 or e_1 are not proper encryptions of elements of D_Q. This is sufficient in the semi-honest model; we discuss methods of handling malicious behaviour in Sect. 4.5.1.

4.4 The GT-CEM Construction and Protocols

Our construction builds on the ideas of the GT protocol (Construction 1 of Chapter 3. This protocol can be cast as a variant of GT-CEM, where one of the inputs is given in plaintext. We present its main idea and observe that a part of this protocol – the randomization procedure – requires S to know his input. In Sect. 4.4.2, we discuss the necessary properties of our new randomization, which works with encryptions only. In Sect. 4.4.3, we present such a randomization procedure and in Sect. 4.4.4 we give a GT-CEM construction. We give an alternative randomization procedure in Sect. 4.4.5, which can be incorporated into our GT-CEM.

4.4.1 The GT Protocol of Chapter 3

For convenience, we give a brief overview of the protocol (Construction 1 of Chapter 3) and emphasize its relevant aspects. Recall, there are two players, a receiver R with input x and a sender S with input y, s_0, s_1. S needs to send R the secret $s_{GT(x,y)}$.

The protocol operates on (homomorphically encrypted) bits of the inputs. The idea is to isolate the "important" position – the one where input bit strings first differ – by mapping it to a predetermined value and simultaneously randomizing values in all other positions. The rest is easily accomplished by applications of linear functions. In this work, we pay special attention to and improve the isolating randomization procedure.

In Construction 1, the Receiver R sends bitwise additively homomorphic encryption of his input $x = \langle x_1, ..., x_n \rangle$ to the Sender S. For each bit position i, S computes (an encryption of) $f_i = x_i \oplus y_i$, i.e. whether $x_i = y_i$. S computes f_i as $x_i - 2x_i y_i + y_i$; note that this requires the knowledge of y_i; knowing $Enc(y_i)$ is not sufficient. It is easy to see that $GT(x, y) = x_j$, where $j = min_{f_i \neq 0} i$. Recall, S's randomization procedure of Construction 1 crucially relies on the fact that $f_j = 1$. In contrast, the randomization procedure of this chapter relies on the (encrypted) difference vector $d_i = x_i - y_i$, the "important element" of which may be (an encryption of) one of $\{-1, 1\}$.

4.4.2 The Intuition of GT-CEM and the Formalization of the Randomization Requirements

Recall, we are given secrets s_0, s_1 and bitwise encryptions of inputs x and y. We can compute an encryption of the bit difference vector d, where $d_i = x_i - y_i$. Elements of the difference vector d assume one of $\{-1, 0, 1\}$. Let $j = min_{d_i \neq 0} i$ be the index of the "important" position. Our goal is to isolate the value d_j by computing an encryption of vector μ, such that $\forall i \neq j, \mu_i \in_R D_{P_{pk}}$ and $\mu_j = d_j$. As in Construction 1, we can obtain such μ_i for $i \geq j$ by computing for $i = 1..n$: $\mu_0 = 0; \mu_i = r_i \mu_{i-1} + d_i$, where $r_i \in_R D_{P_{pk}}$. Now vector μ is a vector of encryptions of (in order): one or more 0, either a 1 or a -1, one or more random elements of $D_{P_{pk}}$. We need to map the zeros of μ to random elements in $D_{P_{pk}}$, while preserving the properties of $\mu_i, i \geq j$. Our randomization maps $-1 \rightarrow s_0, 1 \rightarrow s_1$ (under encryption). At the same time, it maps 0 and random elements from $D_{P_{pk}}$ to random elements from $D_{P_{pk}}$. It is not hard to see (and we explicitly show it in Sect. 4.4.4) that such randomization naturally leads to a GT-CEM.

We believe that such randomization may be useful in other applications as well. Therefore, we formalize its requirements. We present the definition in a slightly more general way, by allowing arbitrary constants instead of $-1, 1$. Further natural extensions of this definition are possible.

Let $v_0, v_1 \in \mathbb{Z} \backslash \{0\}$ be fixed, and $v_0 \neq v_1$. Let $E, \nu, sk, pk, E_{pk}, D_{P_{pk}}, D_{R_{pk}}, D_S$ be as in

Def. 13. Let $i \in \{0,1\}$. We view v_i as an element of $D_{P_{pk}}$ in the natural manner (i.e. as $v_i \mod |D_{P_{pk}}|$). We note that even though this representation may vary with the choice of pk, v_i is a constant. Further, we require $v_i \neq 0 \mod |D_{P_{pk}}|$ and $v_0 \neq v_1 \mod |D_{P_{pk}}|$.

Definition 14. *((v_0, v_1)-Randomizing Mapping) A (v_0, v_1) -* Randomizing Mapping *(RM) is a pair of polytime algorithms $(Rmap, Rec)$ (with implicitly defined domain of mappings $D_{M_{pk}}$), such that the following holds.*

The probabilistic randomized mapping algorithm Rmap takes as input (s_0, s_1, e, pk), where e is an encryption under E_{pk}, and $s_0, s_1 \in D_S$. Rmap outputs an element from $D_{M_{pk}}$. The deterministic recovery algorithm Rec takes as input secret key sk and an element from $D_{M_{pk}}$ and outputs an element from the domain of secrets D_S or a failure symbol $\perp$.

Rmap and Rec satisfy the following conditions:

- *(correctness)* $\forall(sk, pk) \leftarrow Gen(\nu), \forall i \in \{0,1\}, \forall s_0, s_1 \in D_S, \forall \alpha \in D_{R_{pk}}$,
  for $x \in_R D_{P_{pk}}$, *with overwhelming probability in* ν:
  $Rec(Rmap(s_0, s_1, Enc_{pk,\alpha}(v_i), pk), sk) = s_i$
  $Rec(Rmap(s_0, s_1, Enc_{pk,\alpha}(x), pk), sk) = \perp$,
  where the probability is taken over choices of x *and random inputs of Rmap.*

- *(statistical privacy at v_0, v_1)* $\exists Sim$, *s.t.* $\forall(sk, pk) \leftarrow Gen(\nu), \forall s_0, s_1 \in D_S$,
  $\forall i \in \{0,1\}, \forall \alpha \in D_{R_{pk}}$: *the statistical distance*
  $Dist(Sim(s_i, pk), Rmap(s_0, s_1, Enc_{pk,\alpha}(v_i), pk))$ *is negligible in* ν.

- *(statistical privacy at 0 and at random elements of $D_{P_{pk}}$)* $\exists Sim_0$, *such that*
  $\forall(sk, pk) \leftarrow Gen(\nu), \forall s_0, s_1 \in D_S, \forall \alpha \in D_{R_{pk}}$: *the statistical distances*
  $Dist(Sim_0(pk), Rmap(s_0, s_1, Enc_{pk,\alpha}(0), pk))$ *and*
  $Dist(Sim_0(pk), Rmap(s_0, s_1, Enc_{pk,\alpha}(R), pk))$
  are negligible in ν, *where* R *is uniform on* $D_{P_{pk}}$.

We note that a stronger definition might require the last property to hold everywhere, other than at v_0 and v_1. We choose not to include this, since the stronger notion is not necessary for our protocols, and, moreover, Construction 5 would not satisfy it.

4.4.3 A space-efficient $(-1, 1)$-RM

We present a construction for $(-1, 1)$-RM, based on the Paillier encryption scheme [78], which we use to construct GT-CEM. Let E be the Paillier scheme initialized as described in Def. 14. Let *Rmap* be given an encryption under E_{pk}. Our $(-1, 1)$-RM is space optimal in the sense that *Rmap* outputs a single encryption under E_{pk}.

At first glance, the requirements on *Rmap* are conflicting: we must satisfy three data points $((v_0, s_0), (v_1, s_1), (0, \mathit{random}))$ with a linear function (only linear functions can be applied under the homomorphic encryption). Our idea is for *Rmap* to produce not encryptions of secrets s_i, but of their *randomized encodings* S_i. We carefully randomize the encodings S_i, such that their linear combination of interest (i.e. the value that 0 is mapped to) is a random element in $D_{P_{pk}}$.

Let $f = ax + b$ be a linear mapping, such that $f(-1) = -a + b = S_0$ and $f(1) = a + b = S_1$. Then $b = (S_0 + S_1)/2$ and $a = S_1 - (S_0 + S_1)/2 = (S_1 - S_0)/2$. We want to ensure that $f(0) = b = (S_0 + S_1)/2$ is random, while, for $i \in \{0, 1\}$, S_i encodes s_i and contains no other information.

Construction 5. *($(-1, 1)$-RM)*

Let λ and ν be the correctness and security parameters. Let the plaintext group of E_{pk} be $D_{P_{pk}} = \mathbb{Z}_N$, where $N = pq$ is of bit size $n > \nu$. Let $k = \lfloor (n-1)/2 \rfloor$. Define the domain of secrets to be $D_S = D_{S_{pk}} = \{0, 1\}^{k-\lambda}$, and the domain of mappings $D_{M_{pk}}$ to be the domain of encryptions under E_{pk}.

Rmap on input (s_0, s_1, e, pk) proceeds as follows. Set $s'_i = s_i 0^\lambda$ (to help distinguish secrets from random strings). View s'_0, s'_1 as elements of $\mathbb{Z}_N$. Choose $R \in_R \mathbb{Z}_N$ and a bit

$c \in_R \{0,1\}$. Let r_1 (resp. r_0) be the integer represented by k lower (resp. remaining) bits of R, i.e. $R = r_0 2^k + r_1$.

Set S_0, S_1 as follows. If $c = 0$, then set $S_0 = r_0 2^k + s'_0$ and $S_1 = s'_1 2^k + r_1$. If $c = 1$, then set $S_0 = s'_0 2^k + r_1$ and $S_1 = r_0 2^k + s'_1$.

Compute $a = (S_1 - S_0)/2 \mod N$ and $b = (S_0 + S_1)/2 \mod N$.

Finally, apply $f = ax + b$ to e under the encryption and re-randomize the result, that is, choose $r' \in_R \mathbb{Z}_N^*$ and output $e^a g^b r'^N \mod N^2$.

Rec on input (e', sk) proceeds as follows. Rec computes $d = Dec_{sk}(e')$. Let $d_n, ..., d_1$ be the bit representation of d. Let $D_1 = d_{2k}, ..., d_k$ and $D_0 = d_k, ..., d_1$. For $i \in \{0,1\}$, if $D_i = s0^\lambda$, output s and halt. Otherwise output $\perp$.

Theorem 5. *$(Rmap, Rec)$ described in Construction 5 is a $(-1,1)$-RM.*

Proof. We first show that the two correctness properties hold. It is easy to follow the construction of S_i and observe that either its lower k bits or the remaining bits contain the intended secret s_i. Further, the part of S_i that does not represent the secret is random. Therefore the secret is easily distinguishable thanks to the added trailing zeros. Thus, the first correctness condition holds with overwhelming probability in λ. Further, f applied by *Rmap* is a linear function, which is a permutation on $\mathbb{Z}_N$ with overwhelming probability in ν. (Indeed $f = ax + b$ is not a permutation only if $a = (S_1 - S_0)/2$ is not invertible.) Therefore, *Rmap*, evaluated on an encryption of a random element of $\mathbb{Z}_N$, produces a random encryption of a random element of $\mathbb{Z}_N$. It is easy to see that *Rec* outputs $\perp$ on an encryption of a random element with overwhelming probability in λ.

The privacy at v_0, v_1 condition also holds. Indeed, given a secret $s \in D_S$, and pk, the required $Sim(s, pk)$ simulates the output of $Rmap(s_0, s_1, Enc_{pk,\alpha}(v_i), pk)$ as follows. Choose a random bit $c' \in_R \{0,1\}$ and a random $S' \in \mathbb{Z}_N$. If $c' = 0$ set the lower k bits of S' to be $s0^\lambda$. If $c' = 1$ set the the higher $n - k$ bits of S' to be $s0^\lambda$. Return a random encryption of S' under pk. It is easy to see that Sim satisfies the necessary conditions.

The privacy at 0 and at random elements of $\mathbb{Z}_N$ holds for the following reasons. Firstly, as shown in the proof of correctness, *Rmap*, evaluated on encryptions of random elements of $\mathbb{Z}_N$, produces random encryptions of random elements of $\mathbb{Z}_N$. This is easy to simulate with only knowing *pk*. It remains to show that *Rmap* evaluated on an encryption of 0 does the same. Recall, *Rmap* applies f to the input encryption. There are two cases.

If $c = 0$ then $f(0) = 1/2(S_0 + S_1) = 1/2(r_0 2^k + s_0 + s_1 2^k + r_1) = 1/2(r_0 2^k + r_1 + s_0 + s_1 2^k) = 1/2(R + s_0 + s_1 2^k)$.

If $c = 1$ then $f(0) = 1/2(S_0 + S_1) = 1/2(s_0 2^k + r_1 + r_0 2^k + s_1) = 1/2(r_0 2^k + r_1 + s_1 + s_0 2^k) = 1/2(R + s_1 + s_0 2^k)$.

In any case, $f(0)$ is random on $\mathbb{Z}_N$ due to the additive random term $R/2$. □

4.4.4 GT-CEM Based on Bitwise Paillier Encryption of Inputs

Let n be the length of the compared numbers. We will use the Paillier encryption scheme E to encrypt inputs to *Rmap* in the bitwise manner. That is, $Gen(\nu)$ is run, fixing (sk, pk) and the instance E_{pk}. The inputs to *Rmap* are (s_0, s_1, e_0, e_1, pk), where $e_0 = \langle Enc_{pk}(x_1), ..., Enc_{pk}(x_n)\rangle, e_1 = \langle Enc_{pk}(y_1), ..., Enc_{pk}(y_n)\rangle$, where x_1 and y_1 are the most significant bits. The sender additionally has the secrets $s_0, s_1 \in D_S$ as inputs. Let $(Rmap_1, Rec_1)$ be a $(-1, 1)$-RM based on the Paillier encryption scheme (e.g. Constr. 5), instantiated with E_{pk}. Let $D_{M_{pk1}}, D_{S_1}$ be the domains of mappings and secrets of $(Rmap_1, Rec_1)$.

Construction 6. *(GT-CEM)*

Let λ and ν be the correctness and security parameters. Let the plaintext group of E_{pk} be $D_{P_{pk}} = \mathbb{Z}_N$, where $N = pq$ is of bit size $n > \nu$. Define the domain of secrets $D_S = D_{S_1}$ and the domain of mappings $D_{M_{pk}} = D^n_{M_{pk1}}$.

Rmap on input (s_0, s_1, e_0, e_1, pk) computes, for each $i = 1..n$:

1. *an encryption of the difference vector d, where $d_i = x_i - y_i$.*

2. *an encryption of vector γ, s.t. $\gamma_0 = 0$ and $\gamma_i = r_i\gamma_{i-1} + d_i$, where $r_i \in_R \mathbb{Z}_N$.*

3. *a randomized mapping vector μ, where $\mu_i = Rmap_1(s_0, s_1, Enc_{pk}(\gamma_i))$.*

Rmap outputs a random permutation $\pi(\mu)$.

Rec on input $(\mu'_1..\mu'_n, sk)$ proceeds as follows. For $i = 1..n$, let $z_i = Rec_1(\mu'_i, sk)$. If $z_i \neq \perp$, output z_i and halt. Otherwise, if $\forall i = 1..n, z_i = \perp$, output $\perp$.

Theorem 6. *Construction 6 is a GT-CEM.*

Proof. We will first show that Construction 6 satisfies the correctness requirement. It is easy to see that the homomorphic properties of the encryption scheme allow *Rmap* and *Rec* to perform all necessary operations.

Let j be the position where x and y first differ; thus d_j determines $\mathrm{GT}(x, y)$. With overwhelming probability, γ is a vector with the following structure: it starts with zero or more zeros, then, in position j, a one or a minus one, then a sequence of random elements in $\mathbb{Z}_N$. It is not hard to see that, by the correctness and privacy properties of $(-1, 1)$-RM, *Rec*, using Rec_1, will recover $s_{GT(x,y)}$.

We now show that the privacy condition holds as well. We construct simulator $\mathrm{Sim}_{GT}(s, pk)$, where pk is the public key established in the setup phase and $s = s_{Q(x,y)}$. $\mathrm{Sim}_{GT}(s, pk)$ has to generate a distribution statistically close to the output of *Rmap*. $\mathrm{Sim}_{GT}(pk, s)$ proceeds as follows, using the simulators Sim_0 and Sim, required by $(-1, 1)$-RM. It runs $\mathrm{Sim}_0(pk)$ $n-1$ times and $\mathrm{Sim}(s, pk)$ once, obtaining a vector z' of n simulated mappings. $\mathrm{Sim}_R(s, pk)$ outputs a random permutation $\pi'(z')$. It is easy to see that $\mathrm{Sim}_{GT}(pk, s)$ statistically simulates the output of *Rmap*, due to properties of Sim_0 and Sim. □

4.4.5 A General (v_0, v_1)-RM Construction

We informally present the construction for any two constants v_0, v_1. We note that it can be naturally generalized for any number of constants $v_1, ..., v_n$.

Rmap proceeds as follows. First, as in Construction 5, add trailing zeros to s_0, s_1 to distinguish them from random elements in $D_{P_{pk}}$. For $i = 1..2$ do the following. Choose random linear functions $f_i = a_i x + b_i$ on the plaintext domain $D_{P_{pk}}$ of the underlying (Paillier) encryption, such that $f_i(v_i) = s_i$. Apply f_i to the encrypted input, obtaining $Enc_{pk}(s_i)$ if $x = v_i$, or an encryption of a random value otherwise. Re-randomize and randomly permute the two obtained encryptions. It is easy to see that this sequence encodes at most a single secret s_i and contains no other information. *Rec* decrypts the vector, recognizes the secret and outputs it with overwhelming probability.

This (v_0, v_1)-RM can be used with Construction 6, producing GT-CEM with slightly different performance properties. Because this (v_0, v_1)-RM uses larger domains of mappings $D_{M_{pk}}$ than Construction 5, the resulting GT-CEM is less efficient for transferring smaller secrets. When the transferred secrets are large, this (v_0, v_1)-RM performs better due to slightly smaller loss in bandwidth due to redundancy in secrets. See Table in Sect. 4.6 for detailed comparisons.

4.4.6 Resource Analysis

We evaluate the message and modular multiplication efficiency of Construction 6, used with $(-1, 1)$-RM of Sect. 4.4.3 (which we refer to as CEM1) and of Sect. 4.4.5 (CEM2). The generated encryption key is reused for a polynomial number of executions of our protocols, thus we do not count the relatively small computational cost of key generation. Let n be the length of inputs x and y in base 2, and N be the size of the plaintext domain of the Paillier scheme. Then the message complexity (the size of the output of *Rmap*) of CEM1 is $l_1 = n \log(N^2) = 2n \log N$ bits, and that of CEM2 is $l_2 = 2n \log(N^2) = 4n \log N$. We do not count the encrypted inputs x, y for message complexity, since their length is usually small, and, in many settings, they are not sent to S, but computed by S.

To encrypt the $2n$ input bits, $2n \log N$ multiplications are required. Step 1 of Construction 6 requires n multiplications, and step 2 requires $(\log N + 1)n$ multiplications.

Step 3 of CEM1 requires $(3 \log N + 2)n$ multiplications ($2 \log N + 1$ multiplications for application of the linear function f, and $\log N$ to re-randomize the encryption). Similarly, step 3 of CEM2 requires $(6 \log N + 4)n$ multiplications.

Rec of CEM1 (resp. CEM2) costs $2n \log N$ (resp. $4n \log N$) multiplications (We expect to perform half of them before *Rec* recovers the secret and halts).

In total, CEM1 (resp. CEM2) requires no more than $\approx 8n \log N$ (resp. $\approx 13n \log N$) modular multiplications. Of those, $4n \log N$ (resp. $7n \log N$) are performed by *Rmap*, and $4n \log N$ (resp. $6n \log N$) are spent for encrypting inputs and reconstructing the output. Note that the encryption and re-encryption multiplications can be precomputed once the encryption scheme is initialized.

Our modular multiplications are four times slower than those of [33, 41], since they are performed $\mod N^2$, while the Goldwasser-Micali (GM) multiplications (used in [33, 41]) are $\mod N$.

One execution of CEM1 (resp. CEM2) allows transfers of secrets of size up to $(\log N)/2 - \lambda$ (resp. $\log N - \lambda$) for the same cost.

Care must be taken in choosing appropriate parameters for comparisons of our results with the performance of other schemes, in particular those based on the potentially weaker quadratic residuocity assumption ([33, 41]). Note that in practice no known attack on the Paillier system is better than factoring the modulus N. Clearly, factoring based attacks would also be effective against the GM scheme with the same modulus size. Thus we assume that the security of Paillier and GM schemes with the same size moduli is approximately the same.

The performance comparisons are summarized in the Table in Sect. 4.6.

4.4.7 CEM for any NC^1 Predicate From Homomorphic Encryption

We note that it is possible to construct CEM for any NC^1 predicate Q, using, for example, our information-theoretic abstraction of Yao's garbled circuit of Chapter 5. (Recall, NC^1 is the class of decision problems solvable by polynomial size Boolean circuits of depth $O(\log(n))$, and fan-in 2.) The idea is to assign two specially constructed secrets to each input wire of the (polysize) formula representation of the NC^1 circuit. Here each secret corresponds to one of the two possible wire values. The secrets satisfy the following property: a set of secrets, one for each wire of the circuit, allows us to compute the value of the circuit on the corresponding input, and carries no other information.

It is easy to use the homomorphic encryption properties to allow *Rec* to reconstruct only one appropriate secret for each wire. Combined with the tools discussed in the previous paragraph, this implies CEM for any NC^1 predicate.

4.5 Protocol Constructions from GT-CEM

As mentioned in the discussion of CEM in Sect. 4.3, natural protocol constructions immediately arise from CEM in the semi-honest model. We demonstrate this on a special case of PSPP of [33], where the server S runs the auction with two bidders C_0, C_1. (Our solution can accommodate more bidders, using natural techniques.) As discussed in Sect. 4.2 and [33], in the initialization phase, each of the clients generates and publishes his public key pk_i with S.

The main *selection* phase proceeds as follows. Each client C_i sends to S two encryptions of his input, with his own and with the other client's public keys (i.e. S obtains $Enc_{pk_i}(x_i), Enc_{pk_{1-i}}(x_i)$) from C_i). S applies GT-CEM twice (once under each key) and sends the outputs of *Rmap* to the corresponding C_i for reconstruction. That is, S sends $m_i = Rmap(s_0, s_1, Enc_{pk_i}(x_i), Enc_{pk_i}(x_{1-i}), pk_i)$ to each C_i, who then applies

$Rec(sk_i, m_i)$ and obtains s_1 if his bid is greater and s_0 otherwise. (We note that the receipt of the non-winning s_0 is crucial to hide the rank of the bid of C_i in auctions with more than two parties [33].)

It is easy to see that this protocol is secure in the semi-honest model. Indeed, by the definition of CEM, each m_i contains only the intended secret and no other information. Further, it is not hard to see that computationally-bounded S does not learn anything from seeing semantically secure encryptions of clients' bids (under a natural assumption that the secrets s_0, s_1 are a polytime computable function of the transcript of S's view of execution of the auction and arbitrary information available prior to the key generation phase).

4.5.1 Handling Malicious Behaviours

One of the main reasons for the introduction of the semi-honest facilitator is the simplification and efficiency improvement of protocols. In this discussion, we assume the presence of such semi-honest S running *Rmap* and discuss methods of protection against malicious behaviour of other participants. We note that the CEM model is well suited for this task, since the malicious actions of parties are limited to improper input submission and reporting of the decoded output.

First, we observe that the free choice of secrets by S is a powerful tool. For example, when sufficiently long secrets are randomly chosen, they may serve as a proof of the value of Q in the evaluated Q-CEM. Indeed, the recipient of s_i is not able to claim $Q(x, y) = 1 - i$, since he cannot obtain s_{1-i}. Further, for example, secrets can contain S's signatures, proving the correctness of reconstruction to anyone.

A harder task is ensuring that malicious players do not gain from submitting contrived inputs to S. Firstly, zero-knowledge (ZK) techniques could be used to ensure players' compliance with the prescribed protocol. This is often computationally expensive and requires either a common random string or an extra round of interaction. There exist light-weight alternatives to ZK, such as conditional disclosures of Aiello, Ishai and Reingold [1] and Laur and Lipmaa [68]. Their idea, well suited for our setting, is to ensure that an improperly formed input will render useless the obtained output of *Rmap*. For example, suppose *Rmap* requires input encryption e to be a Paillier encryption of a bit (i.e. that $Dec(e) \in \{0, 1\}$). We ensure that non-compliant inputs result in garbled output as follows. Let $s_0, s_1 \in D_S$ be inputs to *Rmap*. We choose a random $r \in_R D_S$ and run *Rmap* with secrets $s_0 \oplus r, s_1 \oplus r$. We now only need a CEM procedure that would transfer r iff $Dec(e) \in \{0, 1\}$, which can be easily constructed.

4.5.2 Proxy Selling with a Secret Reserve Price

We sketch how to apply GT-CEM to an interesting variant of a proxy selling task, mentioned in Sect. 4.1. Here, the seller wishes to be offline and delegate selling to the semi-honest S. The seller initializes E_{pk}, publishes pk and sends an encryption $Enc_{pk}(x)$ of his lowest acceptable price (i.e. reserve) to S, who later interacts with buyers as follows. On an encrypted offer $Enc_{pk}(y)$, S replies with $Rmap(s_0, s_1, Enc_{pk}(y), Enc_{pk}(x), pk)$, where s_1 serves as S's certification of the successful buyer (e.g. in a form of a signature), and s_0 is a non-winning (e.g. empty) secret. Thus, successful buyers obtain (an encryption of) the contract, which they later present to the seller.

Combining GT-CEM with the general CEM techniques based on secret representations, described in sect. 4.4.7, allows us to obtain very efficient CEM depending on several GT evaluations. This allows us to proxy sell not only based on a reserve price, but on a price range, delivery date ranges, etc.

4.6 Comparison with Previous Work

We continue the resource analysis of Sect. 4.4.6. Note that the protocols of [33, 41, 68] can be appropriately modified to be cast as GT-CEM. We summarize the cost of comparable modular multiplications and communication of evaluating GT-CEM based on [33, 41, 68] and our constructions CEM1 and CEM2 (i.e. Construction 4.4.4 instantiated with $(-1,1)$-RM of Sect. 4.4.3 and 4.4.5 respectively).

Here c-bit secrets are transferred based on comparison of n-bit numbers. λ and ν are the correctness and security parameters, and $N > 2^{\nu}$ is the modulus of the employed encryption scheme (GM for [33, 41] and Paillier for [68] and our work). We do not include the one-time cost of key generation. We measure communication as the size of the output of *Rmap*.

Solutions of [33, 41] transfer one-bit secrets per execution, therefore c-bit secrets can be transferred at a factor c cost increase. Our CEM1 (resp. CEM2) protocols transfer secrets of size $c < \nu/2 - \lambda$ (resp. $c < \nu - \lambda$) per execution. Today's common parameters $\nu \approx 1000, \lambda \approx 40..80$ imply transfers of approximately 450 (resp. 950)-bit secrets per execution of CEM1 (resp. CEM2). For CEM of longer secrets, multiple execution is needed. Note the significant advantage of CEM1 for the most frequent case where the transfer of medium-size secrets is required.

Costs and Comparisons. GT-COT of [68] can be modified to obtain GT-CEM similar in cost to CEM2. The solution of Chapter 3 (in a more restricted setting, where one of the compared numbers is given in plaintext) carries approximately half of the cost of CEM2. Other costs and comparisons are summarized below. (The cost of (client-run) GM decryption, used in [41, 33], is not less than $\log N$ modular multiplications. For simplicity, we assume that it is $\log N$.)

Protocol	Comparable Modular Multiplications			Communication	Comment
	client	server	total		
of [41]	$4nc\lambda \log N$	$24nc\lambda$	$24nc\lambda + 4nc\lambda \log N$	$4nc\lambda \log N$	
of [33]	$8n^2c \log N$	$12n^2c$	$12n^2c + 8n^2c \log N$	$8n^2c \log N$	
CEM1	$16n \log N$	$16n \log N$	$32n \log N$	$2n \log N$	$c < \nu/2 - \lambda$
CEM2	$24n \log N$	$28n \log N$	$52n \log N$	$4n \log N$	$c < \nu - \lambda$

Chapter 5

Information Theoretically Secure Formula Evaluation

5.1 Introduction

Summary of the contributions of the chapter. We propose *Gate Evaluation Secret Sharing* (GESS) – a new kind of secret sharing, designed for use in secure function evaluation (SFE) with minimal interaction. The resulting simple and powerful GESS approach to SFE is a generalization of Yao's garbled circuit technique.

We give efficient GESS schemes for evaluating binary gates and prove (almost) matching lower bounds. We give a more efficient information-theoretic reduction of SFE of a boolean formula F to oblivious transfer. Its complexity is $\approx \sum d_i^2$, where d_i is the depth of the i-th leaf of F.

5.1.1 Motivation of the Problem and the Setting

We continue our study of one-round SFE. In this chapter, we give a generic construction for secure evaluation of any NC^1 circuit. (Recall, NC^1 is the class of decision problems solvable by polynomial size Boolean circuits of depth $O(\log(n))$, and fan-in 2.) We approach the problem in a general way by unconditionally reducing SFE to oblivious transfer (OT). OT is a powerful primitive, and is the subject of a vast amount of research. It has been studied in many settings; for example, OT is instantiable with information-theoretic (IT) security (e.g. with noisy channels or a distributed sender [74]). Our SFE constructions automatically apply to all of the above (and many other) settings and will benefit from future OT research. For a discussion of the subtleties, importance and benefits of our setting, see Sect. 2.3 and 2.4.

5.1.2 Our Contributions and Outline of the Work

Our main idea is a new *simple* way of evaluating circuit gates securely by using a new type of secret sharing, which we call *Gate Evaluation Secret Sharing* (GESS). Our method can be viewed as a generalization of Yao's garbled gate evaluation procedure, offering a simple and powerful approach for designing efficient SFE protocols. Our method is flexible, and not limited to $\vee, \wedge, \neg$ gates. Circuits with special purpose (e.g. non-binary) gates may be designed and implemented via GESS to achieve better efficiency for specific functions (see, e.g., Sect. 5.2.6).

We show how a composition of GESS schemes can be used to efficiently reduce SFE to (parallel executions of) 1-out of-2 OT. Given a boolean formula, we obtain a *one-round* reduction, meaning that an instantiation of OT results in a SFE protocol, the security and round complexity of which are that of the underlying OT. Our reduction is very efficient. Previous approaches in part suffer from the exponential (in depth) cost of evaluation of a gate, which has intuitively appeared necessary. We break this intuition by providing a scheme for gate evaluation whose cost is only *quadratic* in the depth of the gate. Further, in our reduction, we don't "pay" for the internal gates of the formula. For a depth d circuit, this results in a factor of approximately $2^{O(\sqrt{d})}$ improvement over previous solutions: $O(2^d d^2)$ vs $\Theta(2^d 2^{\Theta(\sqrt{d})})$. (Like all other approaches, ours suffers from the fact that the number of gates may be exponential in depth. Thus, we offer polytime reduction of only NC^1 circuits.) We prove non-trivial lower bounds, showing that our constructions are almost optimal in the GESS framework.

The GESS approach is especially efficient on small circuits, since it does not use encryption. In Sect. 5.2.6, we demonstrate this by a new efficient protocol for the Two Millionaires problem. This protocol also serves as an example of designing and implementing custom GESS gates.

We start with describing previous approaches and giving conceptual and performance comparisons to our work (Sect. 5.1.3). We then present intuition for our approach and introduce the necessary formal definitions in Sect. 5.2 and 5.2.1. We present our constructions, lower bounds and performance analysis in Sect. 5.2.3 – 5.2.5. In Sect. 5.2.6 we present a new solution of the Two Millionaires problem.

In Sect. 5.3, we show how to use GESS to allow polytime SFE of polysize circuits, when Alice (the party who sends the first message) is polytime. In effect, we obtain another implementation of Yao's garbled circuit approach for the model with polytime Alice, offering essentially the same computational and communication complexity as its best implementations. The natural and efficient handling of the computational setting demonstrates the generality of the GESS approach. We mention that the efficiency of Yao's garbled circuit technique in the standard model can be (slightly) improved by using IT GESS on "the bottom part" of the circuit (see discussion in Sect. 5.3).

We note that, for the ease of understanding, in the main body of this chapter, we present special cases of definitions and proofs. This is, however, sufficient to maintain a high level of formalism in discussion and formally present the underlying ideas. General definitions and more formal proofs are delayed until Sect. 5.4. We stress that the material of Sect. 5.4 is but a relatively simple (but detailed and technical) formalization and generalization of the ideas of this chapter.

5.1.3 Comparisons with Related Previous Work

General discussion. Note the frequent use of a variety of secret sharing schemes in the area of secure function evaluation. They are always used, however, to share secrets *among players*. We contrast this with our novel use, where secrets are shared *among wires* and given to the player who performs reconstruction.

We note that some of the previous approaches (e.g. [29, 60, 61, 63]) are applicable to more general representations of functions (e.g. by arithmetic formulas or branching programs (BP)). Many functions may have especially efficient representations when not restricted to boolean formulas (the setting we consider); secure evaluation of such functions may not benefit from our constructions.

Although our reductions are efficient for polysize boolean formulas of arbitrary depth, they perform better on balanced formulas. For the latter, the complexity is quasi-linear (vs. cubic for highly unbalanced formulas) in the size of the formula. Note that it is possible ([19, 16]) to rebalance any formula to obtain an equivalent log-depth balanced formula, at the cost of small increase in its size (see end of Sect. 5.2.4 for more discussion).

Therefore, for the remainder of this section, assume that we are given a boolean formula (or an NC^1 circuit, which can be viewed as one), which is rebalanced if it benefits the approach considered.

Let d be the depth of the formula or the circuit.

Comparing our reduction to previous constant-round approaches.

Kilian [63] was the first to show a one-round IT reduction (of complexity $\Theta(4^d)$) of SFE to OT. Kilian relies on Barrington's [4] representation of NC^1 circuits as permutation BPs. It is possible to replace Barrington's representation in Kilian's construction with a more efficient construction of Cleve [29] (see, e.g. Cramer et al. [30]). The resulting complexity is $\Theta(2^d 2^{\Theta(\sqrt{d})})$, which is the best previously known for NC^1 circuits and (re)balanced formulas.

Ishai and Kushilevitz [60, 61] suggested a way of representing a circuit as a predicate on a vector of degree 3 (degree of the input variables x_i is 1) *randomizing polynomials*. Their construction assigns an (exponential in d in size) polynomial representation to each wire of the corresponding fan-out 1 circuit, and implies a one-round SFE-to-OT reduction, of complexity $\Theta(4^d)$. They also previously suggested a related *Private Simultaneous Messages* (PSM) model [59] of computation. They showed how to evaluate functions computed by BPs in the PSM model (and also in our SFE-to-OT reduction model) with resources quadratic in the size of the BP. (Recall, BPs are more powerful than permutation BPs or formulas.) For our setting, their approach implies a one-round SFE-to-OT reduction of cost $\Theta(4^d)$, using an (almost) linear in size transformation of a formula to a BP [47].

Our reduction of boolean formulas is simpler and more efficient (costing $O(2^d d^2)$) than the above approaches.

Yao's garbled circuit approach can also be used for such reduction (see, e.g. [61]). The idea is to use an IT-secure encryption scheme (e.g. using one-time pad) in Yao's garbled circuit. The keys of such a scheme must be more than twice the size of the secret, causing an exponential (in d) growth of the size of secrets, even in fan-in 1 circuits[1]. The complexity of such a scheme is about $\Theta(4^d)$ (up to 2^d leaves, each of size up to 2^d). Our approach is a generalization and an improvement of this approach.

Sander, Young and Yung (SYY) ([86]) present a "fully homomorphic" encryption scheme and apply it to SFE. The encryption size grows exponentially with the number of the applied OR operations, resulting in $\Theta(8^d)$ cost of SFE. Beaver [6] suggests an optimization of the SYY pyramid and extends the approach to the multi-party setting, achieving complexity $\Theta(4^d)$. Further, using the representation of Feige, Kilian and Naor [38] of NLOGSPACE as a product of polysize matrices, he shows how to compute it in one round, bootstrapping the SYY approach, also achieving complexity $\Theta(4^d)$. Our approach

[1]Note the distinction between this flavour of Yao's approach and its standard version for evaluation of polysize circuits (e.g. [5, 85, 76, 70]). The latter is not a reduction to OT; e.g, it cannot be used to construct one-round protocols IT-secure against Alice.

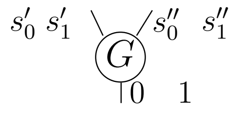
is conceptually different, simpler, more composable, uses fewer assumptions, and offers complexity of at most $O(2^d d^2)$. Also, unlike SYY, we do not have the requirement of a layered circuit, which further increases our performance improvement.

5.1.4 Our Setting

We are working in a setting with two semi-honest participants who use randomness in their computation. A large part of our work concerns the reductions of various problems to the OT oracle. In the semi-honest model, secure reductions result in secure protocols when the called oracles are replaced by their secure implementations. Further, the oracles' implementations may be run in parallel, which, with natural OT implementations, results in secure one-round protocols. See Goldreich [49] for definitions, discussion and the composition theorem.

5.2 The GESS Approach

The intuition behind the GESS approach. Suppose first that the circuit C consists of a single binary gate G with two inputs, one held by Alice, and one by Bob. To transfer the value of the output wire to Alice, Bob encodes possible values of each of the two input wires and transfers to Alice two of the four encodings – one for each wire. Encoding of Alice's wire value is sent via OT. Each pair of encodings that can be possibly sent, has to allow the recovery of the corresponding (to G) value of the output wire, and cannot carry any other useful information. Consider the following example.



Given the possible output values $0, 1$ and the semantics of the gate G, Bob generates encodings of the input wires' values $(s'_0, s'_1), (s''_0, s''_1)$, such that each possible pair of encodings s'_i, s''_j, where $i, j \in \{0, 1\}$, allows to reconstruct $G(i, j)$, and carries no other information. Now, if Bob sends Alice shares corresponding to their inputs, Alice would be able to reconstruct the value of the output wire, and nothing else.

This mostly corresponds to our intuition of secret sharing schemes. Indeed, the possible gate outputs play the role of secrets, which are shared and then reconstructed from the input wires encodings (shares).

Our next observation is that Bob need not share the *values* of the output wire, but instead can share their *encodings*, which, in turn, may be input shares of another gate. Thus, Alice and Bob can recursively apply the GESS approach to multi-gate circuits. For each wire, Alice will only be able to obtain one secret – the one corresponding the the value of the wire on the parties' inputs.

Relationship to Yao's garbled circuit approach. As briefly mentioned above, our protocols can be easily modified into Yao's garbled circuit procedure. See Sect. 5.3 for a construction and proof of security. The construction of Sect. 5.3 additionally provides intuition on the relationship between our contributions and Yao's solution. We remind the reader of the main difference – the standard Yao's construction relies on computational assumptions (private-key encryption), and thus is not secure against computationally unbounded adversaries.

5.2.1 The Definition of Gate Evaluation Secret Sharing

We now formally state the desired properties of the secret sharing scheme. While the idea of the definition is quite simple, it is somewhat burdened with notation due to the necessary level of formalism. For simplicity, we present the definition for the case of a gate with two binary inputs and a binary output, postponing the presentation of its most general form to Sect. 5.4.1 (Def. 16). A simple instructive example of a GESS scheme is Constr. 8 in Sect. 5.2.3.

Let G be a gate with two binary inputs and a binary output. Also denote by $G : \{0,1\} \times \{0,1\} \mapsto \{0,1\}$ the function computed by gate G. Let SEC be the domain of secrets. Suppose we've associated a secret $s_i \in SEC$ with each of the two possible values i of the output wire of G. In general, distributions of s_0 and s_1 may be dependent, so we talk about a *tuple* of secrets $\langle s_0, s_1 \rangle$ from a domain of tuples $TSEC \subset SEC^2$ associated with the output wire. We want to assign a share to each value of the two input wires, such that each combination of shares allows reconstruction of (only) the "right" secret. As do secrets, shares on a wire form a tuple: $\langle sh_{10}, sh_{11} \rangle \in TSH_1 \subset (SH_1)^2$ on wire 1, and $\langle sh_{20}, sh_{21} \rangle \in TSH_2 \subset (SH_2)^2$ on wire 2. In our notation, $sh_{ij} \in SH_i$ is the share of the i-th input wire ($i \in \{1, 2\}$), corresponding to the value $j \in \{0, 1\}$.

Definition 15. *(Gate evaluation Secret Sharing) A* gate evaluation *secret sharing scheme (*GESS*) for evaluating G as above (we also say GESS* implementing *G) is a pair of algorithms (Shr, Rec) (with implicitly defined secrets domain SEC, secrets tuples domain $TSEC$, two share domains SH_1 and SH_2 and two share tuples domains TSH_1, TSH_2), such that the following holds.*

The probabilistic share generation algorithm Shr takes as input a two-tuple of secrets $\langle s_0, s_1 \rangle \in TSEC$ and outputs two tuples of shares (one for each wire), where, $\forall i \in \{1, 2\}$, the i-th tuple $t_i \in TSH_i$ consists of two shares $sh_{ij} \in SH_i$. The deterministic share reconstruction algorithm Rec takes as input two elements $sh_1 \in SH_1$ and $sh_2 \in SH_2$ and outputs $s \in SEC$.

Let $v = \langle v_1, v_2 \rangle \in \{0, 1\} \times \{0, 1\}$ be a selection vector. Define the selection function $Sel(\langle sh_{10}, sh_{11} \rangle, \langle sh_{20}, sh_{21} \rangle, v) = \langle sh_{1v_1}, sh_{2v_2} \rangle$. Write $V_1 \equiv V_2$ to denote that V_1 and V_2 are distributed identically.

Shr and Rec satisfy the following conditions:

- *correctness: for all random inputs of Shr and secrets tuples $\langle s_0, s_1 \rangle \in TSEC$, $\forall v \in \{0, 1\}^2, Rec(Sel(Shr(\langle s_0, s_1 \rangle), v)) = s_{G(v)}$*
- *privacy (selected shares contain no information other than the value $s_{G(v)}$): There exists a simulator Sim, such that $\forall \langle s_0, s_1 \rangle \in TSEC$ and any $v \in \{0, 1\}^2$:*

$Sim(s_{G(v)}) \equiv Sel(Shr(\langle s_0, s_1\rangle), v)$

Observation 3. *A simple generalization of this definition (required for discussion in Sect. 5.2.3 and 5.2.4) considers the identity gate G_I with a four-valued output wire, where each output corresponds to a pair of inputs. In this case, the secrets form a 4-tuple $\langle s_{00}, ..., s_{11}\rangle$, while there are still two two-tuples of shares. Note that we can convert GESS implementing G_I into GESS implementing any other binary gate by simply restricting some of the secrets to be equal. Denote the correspondence between a secret $s \in SEC$ and the wire value $v \in \{0,1\}$ by $s \leftrightarrow v$. Then setting $s_{01} = s_{10} = s_{11} \leftrightarrow 1, s_{00} \leftrightarrow 0$ gives the implementation of the OR, and $s_{00} = s_{01} = s_{10} \leftrightarrow 0, s_{11} \leftrightarrow 1$ – of the AND gates. NOT gates can be implemented "for free" by simply eliminating them and inverting the correspondence of the appropriate wire's values and secrets.*

Observation 4. *We note that, in contrast with the traditional approach of multi-secret sharing schemes, our definition allows the possibility that a single share gives out some information about a secret. It is easy to see, however, that for the gates that depend on both inputs, this information must be common to* every *secret, since otherwise it is possible to determine whether a corresponding combination of secret/share occurred, which allows to easily construct a distinguisher breaking the privacy requirement of GESS. Further, shares of the same wire, corresponding to different values, must be distributed identically (otherwise a distinguisher exists).*

The definition is given for specific input and output domains, and therefore we do not talk about polynomial bounds on *Shr* and *Rec*. However, in practice, we are interested in *ensembles* of schemes and want them to be uniform polytime algorithms. Note that in the definition, we don't insist on an ensemble of *efficient* simulators. This is because an efficient simulator exists if any one exists. Indeed, an efficient simulator Sim_{eff} can simply output $Sel(Shr(\langle s_0, s_1\rangle), v)$, where at least one of the secrets s_i is equal to s, and v is any selection vector, such that $G(v) = i$. S_{eff} is as efficient as the *Shr* function. Further, the existence of the (possibly inefficient) simulator guarantees the equality of distributions $Sel(Shr(\langle s_0, s_1\rangle), v)$ for all inputs as above. Therefore the the perfect simulation property of Sim_{eff} holds.

5.2.2 Reduction of SFE to OT using GESS

Suppose Alice and Bob have a circuit C, consisting of fan-out 1 gates $G_1, G_2, ...$ We formally describe a reduction of securely evaluating C on their inputs to calls to OT, resulting in a one-round protocol. Again, for simplicity of presentation we assume that all gates G_i are fan-in 2 binary gates.

Assume that for every gate G of C, there exists a GESS $GESS_G : (Shr_G, Rec_G)$ of Def. 15 with appropriate secret domains (as described below). We give explicit constructions (e.g. Constr. 8 in Sect. 5.2.3) of such schemes for all gates with two binary inputs. We note that GESS for every other gate can be constructed (e.g. from Constr. 7 instantiated with GESS of Constr. 8).

Construction 7. *(Reducing SFE to OT)* **Bob's precomputation.** *Bob starts with the output gate. He sets the secrets domain SEC of it to be $\{0, 1\}$ and sets the secrets tuple to $\langle 0, 1\rangle$. He proceeds through gates of C recursively as follows.*

Consider a gate G. Let $TSEC$ and a secrets tuple $t = \langle s_0, s_1\rangle \in TSEC$ are given for G. Let $GESS_G$ be a GESS scheme implementing G with secrets tuples domain $TSEC \subset SEC^2$. Bob runs Shr_G on the secrets tuple t and obtains two tuples of shares $t_1 \in TSH_1$ and $t_2 \in TSH_2$, corresponding to the first and second input wires of G respectively. Let G'_i be the i-th input gate of G ($i \in \{0, 1\}$). Then Bob processes G'_i as follows. He treats the tuple of shares $t_i \in TSH_i$ of G's input wire as the tuple of secrets of G'_i, and TSH_i – as the secrets tuples domain of G'_i. Bob now applies the algorithm of this paragraph to G'_i.

Eventually, Bob obtains secrets tuples for all input wires of C. Note that Bob's choices of instances of GESS schemes for the gates of C are deterministic and built into the *protocol; this makes explicit the corresponding Rec procedures.*

Interaction. *For each input wire associated with Alice, she and Bob make (parallel) calls to OT oracles. Alice has the wire's input and Bob has the tuple of secrets as their inputs of each of the calls. For each input wire associated with Bob, Bob sends Alice the corresponding secret from that wire's tuple of secrets*[2]*.*

Alice's computation. *Alice obtains results of the OT and the secrets corresponding to Bob's inputs. Alice proceeds, from the top down on the circuit C, as follows. For each gate, Alice knows the secrets corresponding to the inputs of the gate, and the corresponding Rec procedure. She runs Rec on the input secrets and obtains the output secret. She proceeds in this manner until she obtains the secret corresponding to the output wire. Alice outputs this secret.*

Theorem 7. *Constr. 7 is a non-cryptographic reduction (thus unconditionally secure against both Alice and Bob) of SFE of C to OT, in the semi-honest model.*

The proof of Theorem 7 is intuitive and is delayed until Sect. 5.4.2.

Observation 5. *A circuit C with fan-out greater than* 1 *can be converted into a corresponding (potentially very large) tree-circuit C' by duplicating C's subtrees where appropriate. Equivalently, one can view the secrets as being computed and propagated by Bob* in parallel *on the same wire. Note that we, however, need not increase the* number *of corresponding OT instances due to the growth of C' relative to C (until a certain efficiency threshold is reached). Rather, Bob's inputs to OT will be longer (without the increase in the total number of bits transferred). This will often result in significant computational and communication savings.*

[2]This message is appended to Bob's messages of the n-round instantiations of OT oracles to form an n-round protocol.

5.2.3 GESS for gates with two binary inputs

We now present an efficient ensemble of GESS schemes (indexed by the secrets domains) implementing any binary gate with two binary inputs. This construction is a building block of a more efficient Constr. 9. We present GESS for the 1-to-1 gate function $G : \{0,1\}^2 \mapsto \{00, 01, 10, 11\}$, where $G(0,0) = 00, G(0,1) = 01, G(1,0) = 10, G(1,1) = 11$ (see Observation 3 for justification).

Let the secrets domain be $SEC = \{0,1\}^n$, and four (not necessarily distinct) secrets $s_{00}, ...s_{11} \in SEC$ are given; the secret s_{ij} corresponds to the value $G(i,j)$ of the output wire. Note that $|SEC| \geq 4$ need not hold; our scheme is interesting even when $|SEC| \geq 2$.

Intuition of the scheme. The main idea of the design of the GESS scheme is as follows. We first randomly choose two strings $R_0, R_1 \in_R SEC$ to be the shares sh_{10} and sh_{11} (corresponding to 0 and 1 of the first input wire). Now consider sh_{20} – the share corresponding to 0 of the second input wire. We want this share to produce either s_{00} (when combined with sh_{10}) or s_{10} (when combined with sh_{11}). Thus, the share $sh_{20} = B_{00}B_{10}$ will consist of two blocks. One, $B_{00} = s_{00} \oplus R_0$, is designed to be combined with R_0 and reconstruct s_{00}. The other, $B_{10} = s_{10} \oplus R_1$, is designed to be combined with R_1 and reconstruct s_{10}. Share $sh_{21} = B_{01}B_{11}$ is constructed similarly, setting $B_{01} = s_{01} \oplus R_0$ and $B_{11} = s_{11} \oplus R_1$. Note the indexing notation – the secret s_{ij} is always reconstructed using B_{ij}.

Both leftmost blocks B_{00} and B_{01} are designed to be combined with the same share R_0, and both rightmost blocks B_{10} and B_{11} are designed to be combined with R_1. Therefore, we append a 0 to R_0 to tell *Rec* to use the left block of the second share for reconstruction, and append a 1 to R_1 to tell *Rec* to use the right block of the second share for reconstruction. Finally, to hide information leaked by the order of blocks in shares, we perform the following. We randomly choose a bit b; if $b = 1$, we reverse the order of blocks in *both* shares of wire 2 and invert the appended pointer bits of the shares of wire 1. More formally:

Construction 8. *(GESS ensemble for gates with two binary inputs.) Let* $SEC = \{0,1\}^n$ *and* $TSEC = SEC^4$ *be the secrets domains. Let the secrets tuple* $\langle s_{00}, ..., s_{11} \rangle \in TSEC$ *be given. The domains of shares are:* $SH_1 = \{0,1\} \times SEC$ *and* $SH_2 = SEC^2$*. Note that* $TSH_1 = SH_1^2$ *and* $TSH_2 = SH_2^2$*.*

Shr chooses $b \in_R \{0,1\}, R_0, R_1 \in_R SEC$ *and sets blocks* $B_{00} = s_{00} \oplus R_0$*,* $B_{01} = s_{01} \oplus R_0$*,* $B_{10} = s_{10} \oplus R_1$*,* $B_{11} = s_{11} \oplus R_1$*.*

Shr sets the tuples of shares $\langle sh_{10}, sh_{11} \rangle \in SH_1, \langle sh_{20}, sh_{21} \rangle \in SH_2$ *as follows*

	wire 1	*wire* 2*, if* $b = 0$	*wire* 2*, if* $b = 1$
wire value 0	$sh_{10} = bR_0$	$sh_{20} = B_{00}B_{10}$	$sh_{20} = B_{10}B_{00}$
wire value 1	$sh_{11} = \bar{b}R_1$	$sh_{21} = B_{01}B_{11}$	$sh_{21} = B_{11}B_{01}$

Rec proceeds as follows. On input $Sh_1 = b'r$*,* $Sh_2 = a_0a_1$*, Rec outputs* $r \oplus a_{b'}$*.*

Theorem 8. *For each* $n \in \mathbb{N}$*, Constr. 8 is a GESS scheme.*

Proof. To prove correctness, we need to show that no matter what the random choices of Shr and the wire values i_1, i_2 are, Rec always reconstructs $s_{G(i_1,i_2)}$.

For the proof of security, suppose secrets $s_{00}, ..., s_{11}$ are given. This determines the distribution on the Shr generated shares. Let the input wire values i_1, i_2 be given. Then the distribution P on the corresponding pair of shares $\langle sh_{1i_1}, sh_{2i_2} \rangle$ and the secret $s = s_{G(i_1,i_2)}$ shared by the pair are determined. The goal of the simulator is, given only s, to generate a pair of shares distributed identically to P. Note that this exactly corresponds to the privacy condition $Sim(s_{G(i_1,i_2)}) \equiv Sel(Shr(s_{00}, ..., s_{11}), \langle i_1, i_2 \rangle)$ of Def. 15.

The following natural simulator $Sim(s)$ suffices. On input $s \in SEC$, Sim chooses a random bit $d \in_R \{0,1\}$ and random strings $p, q \in_R SEC$. If $d = 0$, he outputs $(\langle d, p \rangle, \langle p \oplus s, q \rangle)$, otherwise he outputs $(\langle d, p \rangle, \langle q, p \oplus s \rangle)$.

Proof of correctness and security is simple and is done by case analysis. We need to consider the four possible combinations of gate input values $i_1, i_2 \in \{0,1\}$. We show correctness and that Sim perfectly simulates the corresponding truly generated shares. Denote random variables $\langle sh_1, sh_2 \rangle = \langle b'r, a_0a_1 \rangle = Sel(Shr(s_{00}, ..., s_{11}), \langle i_1, i_2 \rangle)$. We write out only one case; others are analogous.

Case $i_1 = 0, i_2 = 0$. Thus $s = s_{G(0,0)}$.

Correctness: If $b = 0$, then $b' = 0, sh_1 = 0R_0, sh_2 = (s_{00} \oplus R_0, s_{10} \oplus R_1)$. $Rec(sh_1, sh_2) = R_0 \oplus (s_{00} \oplus R_0) = s_{00} = s$. If $b = 1$, then $b' = 1, sh_1 = 1R_0, sh_2 = (s_{10} \oplus R_1, s_{00} \oplus R_0)$. $Rec(sh_1, sh_2) = R_0 \oplus (s_{00} \oplus R_0) = s_{00} = s$.

Security: Clearly, $Sim(s)$ perfectly simulates sh_1. Further, sh_2 consists of two blocks $B_{00} = s \oplus R_0$ and $B_{10} = s_{10} \oplus R_1$. Observe that $B_{10} = s_{10} \oplus R_1$ is distributed uniformly randomly on SEC (since R_1 is random on SEC and secret). Therefore, sh_2 consists of two blocks from SEC, where one block is random on SEC and the other is equal to $s \oplus R_0$, where the non-random block is pointed by the bit b' of sh_1, Therefore $Sim(s)$ also perfectly simulates sh_2 and the pair $\langle sh_1, sh_2 \rangle$, since d is distributed identically to b'. □

The Permute and Point (PP) Technique. We note the application of the following technique: we permuted the blocks of the shares of the second wire, and appended pointers to the shares of the first wire, hiding information contained in the order of blocks. We use the same idea in all other constructions in this chapter (of Sect. 5.2.4 and 5.2.6). We believe this technique is likely to be useful in many other GESS constructions; it may also have other applications.

Observation 6. *We note that the simulator Sim of Theorem 8 is the same for every gate function – it is only the secrets semantics that defines the semantics of the gate. Therefore, Sim can simulate gates without knowing what they are. Therefore, when this secret sharing scheme is plugged into the protocol of Sect. 5.2.2, semantics of all gates are unconditionally hidden from Alice - she only knows the wire connections of C.*

5.2.4 The Main Construction – GESS for AND/OR/NOT Circuits

Note the inefficiency of Constr. 8, causing the shares corresponding to the second input wire be double the size of the gate's secrets. While, in some circuits, we could avoid the exponential (in depth) secret growth by balancing the direction of greater growth toward more shallow parts of the circuit, a more efficient solution is desirable. We discuss only AND/OR circuits, since NOT gates are given for "free" (see Observation 3).

Recall, in Constr. 8 each of the two shares of the second wire consists of two blocks. Observe that in the case of OR and AND gates either left or right blocks of the two shares are equal. We use this property to reduce (relative to Constr. 8) the size of the shares when the secrets are of the above form. Our key idea is to view the shares of the second wire as being the same, except for one block.

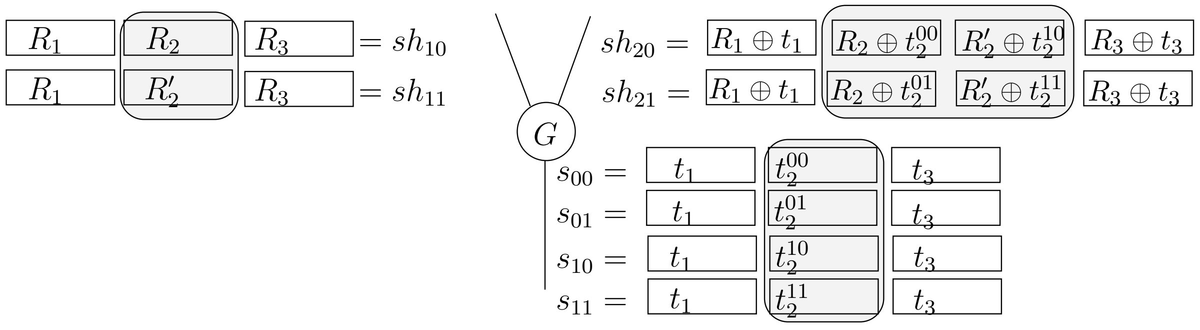
Suppose each of the four secrets consists of n blocks and the secrets differ only in the j^{th} block, as follows:

$$s_{00} = (\quad t_1 \quad \dots \quad t_{j-1} \quad t_j^{00} \quad t_{j+1} \quad \dots \quad t_n \quad), \quad \dots$$
$$s_{11} = (\quad t_1 \quad \dots \quad t_{j-1} \quad t_j^{11} \quad t_{j+1} \quad \dots \quad t_n \quad),$$

where $\forall i = 1..n$: $t_i, t_j^{00}, t_j^{01}, t_j^{10}, t_j^{11} \in D$, for some domain D of size k. It is convenient to consider the *columns* of blocks, spanning across the shares. Every column (with the exception of the j-th) consists of four equal blocks. We stress that the index j is only determined by the secrets, and must not be recovered at reconstruction.We construct a GESS for gates with two binary inputs, where the size of each share of the first wire is $n(k + \lceil\log(n+1)\rceil)$ and of the second wire is $(n+1)k$. Further, each share of the first wire consists of n blocks of size $|D| + \lceil\log(n+1)\rceil$, and all but one pair of corresponding blocks are equal between the shares. Each share of the second wire consists of $n+1$ blocks of size $|D|$ and, for OR and AND gates, all but one pair of corresponding blocks are equal between the shares. Since the generated shares satisfy the above conditions on secrets, repeated application of this GESS for OR and AND gates is possible.

The scheme's intuition. For simplicity of presentation, we do not present the GESS scheme in full generality here (this is postponed to Sect. 5.4.3). We show its main ideas by considering the case where the four secrets consist of $n = 3$ blocks each, and $j = 2$ is the index of the column of distinct blocks.

Our idea is to share the secrets "column-wise", that is to treat each of the three columns of blocks of secrets as a tuple of subsecrets and share this tuple separately, producing the corresponding subshares. Consider sharing the 1-st column. All four subsecrets are equal (to $t_1 \in D$), and we share them trivially by setting both subshares of the first wire to a random string $R_1 \in_R D$, and both subshares of the second wire to be $R_1 \oplus t_1$. Column 3 is shared similarly. We share column 2 as in Constr. 8 (highlighted on the diagram), omitting the last step of appending the pointers and permutation. This preliminary assignment of shares (still leaking information due to order of blocks) is shown on the diagram.



R_1 R_2 R_3 $= sh_{10}$
R_1 R'_2 R_3 $= sh_{11}$
G
$sh_{20} =$ $R_1 \oplus t_1$ $R_2 \oplus t_2^{00}$ $R'_2 \oplus t_2^{10}$ $R_3 \oplus t_3$
$sh_{21} =$ $R_1 \oplus t_1$ $R_2 \oplus t_2^{01}$ $R'_2 \oplus t_2^{11}$ $R_3 \oplus t_3$
$s_{00} =$ t_1 t_2^{00} t_3
$s_{01} =$ t_1 t_2^{01} t_3
$s_{10} =$ t_1 t_2^{10} t_3
$s_{11} =$ t_1 t_2^{11} t_3

Note that the reconstruction of secrets is done by XOR'ing the corresponding blocks of the shares, and, importantly, the procedure is the same for both types of sharing we use. For example, given sh_{10} and sh_{21}, we reconstruct the secret $(R_1 \oplus (R_1 \oplus t_1),\ \ R_2 \oplus (R_2 \oplus t_2^{01}),\ \ R_3 \oplus (R_3 \oplus t_3)) = s_{01}$.

The remaining (PP) step (not shown on the diagram) is to randomly permute the order of the four columns of both shares of wire 2 and to append ($\log 4$)-bit pointers to each block of the shares of wire 1, telling *Rec* which block of the second share to use. Note that the pointers appended to both blocks of column 1 of wire 1 are the same.

The same holds for column 3. Pointers appended to blocks of column 2 are different. For example, if the identity permutation was applied, then we will append "1" to both blocks R_1, "2" to R_2, "3" to R_2', and "4" to both blocks R_3. Because G is either an OR or an AND gate, both tuples of shares maintain the property that all but one pairs of corresponding blocks are equal between the shares of the tuple. Note that it is not a problem that the index of the column with different entries on input wire 1 is the same as that on the output wire: since the adversary never sees both shares of any wire, this index remains unconditionally hidden.

Construction 9. *(GESS for AND/OR gates) The presented construction can be naturally generalized for an arbitrary number of blocks n of size k and for arbitrary index j of the column with differing blocks. The formal presentation of this general construction is postponed to Sect. 5.4.3 (Constr. 12).*

Theorem 9. *For each $n, k, j \in \mathbb{N}$, Constr. 9 is a GESS scheme as defined by (a generalization of) Def. 15.*

We give the intuition of the proof and refer the reader to Sect. 5.4.3 for details. First, the correctness of the reconstruction is easily verifiable. Further, each of the four pairs of shares, reconstructing their corresponding secret $s \in \{s_{00}, .., s_{11}\}$, has the following structure. Let $s = (t_1, ..., t_n)$. The second share in each pair of shares is a sequence of $n+1$ randomly chosen blocks r_i from D: $sh_2 = (r_1, ..., r_{n+1})$. The first share in each pair is a sequence of n "blocks with pointers" $sh_1 = (B_1, ..., B_n)$, as follows. $\forall i \in \{1..n\}, B_i = \langle p_i, b_i \rangle$, where $p_1, ..., p_n$ is a random permutation of a random n-element subset of $\{1..n+1\}$, and $b_i = t_i \oplus r_{p_i} \in D$. This implies the simulator $Sim(s)$, required by Def. 15.

GESS' performance. From above, if the secrets of the output wire of G consist of n blocks of size k, then the secrets of G's inputs consist of no more than $n+1$ blocks of size $k + \lceil \log(n+1) \rceil$. Similarly, d levels deeper, wires' secrets consist of no more than $n + d$ blocks of size $k + \sum_{i=1..d}\lceil\log(n + i)\rceil$. Therefore, starting with one-bit secrets ($n = 1, k = 1$), a tree circuit will have at depth d secrets of size at most $(d + 1)(d \log(d + 1) + 1) = d^2 \log(d + 1) + d \log(d + 1) + d + 1$. The shares grow very slowly: as $d \to \infty$, the "share expansion factor" — the ratio of sizes of shares to sizes of secrets of a GESS scheme for a gate G at depth d — approaches 1. Since the gates have exactly two inputs, there are at most 2^d input wires to the circuit, and the total size of Bob's secrets to be sent to Alice is $2^d(d^2 \log(d + 1) + d \log(d + 1) + d + 1) \approx 2^d d^2 \log d$, dominated by the 2^d term.

Rebalancing C. We note that rebalancing C prior to applying the above reduction may result in substantial performance improvement. Bonet and Buss [16] and Bshouty, Cleve and Eberly [19] prove the following fact (and exhibit the rebalancing procedure).

Let C be a $\{\vee, \wedge, \neg\}$-formula of leaf size m. Then for all $k \geq 2$, there is an equivalent $\{\vee, \wedge, \neg\}$-formula C', such that $\text{depth}(C') \leq (3k \ln 2) \cdot \log m$, and $\text{leafsize}(C') \leq m^\alpha$, where $\alpha = 1 + \frac{1}{1+\log(k-1)}$.

Consider a highly unbalanced C of size m. Direct application of our reduction costs $\Theta(m^3)$, more than BP based approaches [59, 60, 61] of cost $O(m^2)$. Rebalancing C as above, even sub-optimally setting $k = 9$, results in a formula C' of size $m^{1.25}$ and depth $\approx 18.5 \log m$. Applying the reduction to C' yields a much better cost $O(m^{1.25} \log^2 m)$. An optimal (w.r.t. the cost of the GESS reduction) choice of k or better rebalancing will further improve our (but not BP's) performance.

5.2.5 Lower Bounds for GESS – The Optimality of Our Constructions

Let $i, j \in \{0, 1\}$. Denote by A_i (resp. B_i) the random variable of the share corresponding to the wire value i of the first (resp. second) input wire. Denote by S_{ij} the random variable of the secret corresponding to the gate output value $G(i, j)$. Let $H(\cdot)$ be Shannon entropy. We start with proving a technical lemma.

Lemma 1. *For any GESS scheme implementing a gate with binary inputs,* $H(A_i) + H(B_j) \geq H(S_{i(1\text{-}j)}|B_{1\text{-}j}) + H(S_{(1\text{-}i)j}|A_{1\text{-}i}) + H(S_{ij}|S_{i(1\text{-}j)}S_{(1\text{-}i)j}S_{(1\text{-}i)(1\text{-}j)})$.

Proof. For simplicity, prove the lemma for $i = j = 0$, i.e that $H(A_0) + H(B_0) \geq H(S_{01}|B_1) + H(S_{10}|A_1) + H(S_{00}|S_{01}S_{10}S_{11})$. Other cases are analogous.

First, since $H(S_{01}|A_0B_1) = 0$, and using the chain rule twice, obtain $H(A_0|B_1) = H(A_0S_{01}|B_1) - H(S_{01}|A_0B_1) = H(A_0S_{01}|B_1) = H(S_{01}|B_1) + H(A_0|B_1S_{01})$. Similarly, $H(B_0|A_1) = H(S_{10}|A_1) + H(B_0|A_1S_{10})$.

By definition, A_1, B_1 do not reveal anything about S_{00} (other than what's implied by S_{11}), and, further, A_0, B_0 recover S_{00}. Then $H(S_{00}|S_{01}S_{10}S_{11}) \leq H(S_{00}|A_1B_1S_{01}S_{10}) \leq H(A_0B_0|A_1B_1S_{01}S_{10}) \leq H(A_0|A_1B_1S_{01}S_{10}) + H(B_0|A_1B_1S_{01}S_{10}) \leq H(A_0|B_1S_{01}) + H(B_0|A_1S_{10})$.

Thus, $H(A_0) + H(B_0) \geq H(A_0|B_1) + H(B_0|A_1) \geq H(S_{01}|B_1) + H(A_0|B_1S_{01}) + H(S_{10}|A_1) + H(B_0|A_1S_{10}) \geq H(S_{01}|B_1) + H(S_{10}|A_1) + H(S_{00}|S_{01}S_{10}S_{11})$. □

Because all shares corresponding to the same wire must be distributed identically (see Observation 4), their entropies must be equal. Thus Lemma 1 implies that $\forall i_1, i_2 \in \{0,1\} : H(A_{i_1}) + H(B_{i_2}) \geq$
$MAX_{i,j\in\{0,1\}}(H(S_{i(1-j)}|B_{1-j}) + H(S_{(1-i)j}|A_{1-i}) + H(S_{ij}|S_{i(1-j)}S_{(1-i)j}S_{(1-i)(1-j)}))$.

Consider non-trivial gates – those that depend on both (binary) inputs. Note that the gate output need not be binary. We show the optimality of constructions for the natural case when the secrets are drawn independently at random from the same domain (with only the restrictions of secrets equality imposed by the semantics of G). In that case, by Observation 4, $H(S_{i(1-j)}|B_{1-j}) = H(S_{i(1-j)})$ and $H(S_{(1-i)j}|A_{1-i}) = H(S_{(1-i)j})$. Consider the two possible cases.

Case 1: there exist gate inputs i, j, s.t. $G(i, j)$ is not equal to the gate value on any other inputs. This is the case for most non-trivial gates (including AND and OR). In this case, $H(S_{ij}|S_{i(1-j)}S_{(1-i)j}S_{(1-i)(1-j)}) = H(S_{ij})$ and thus $\forall i_1, i_2 \in \{0,1\} : H(A_{i_1}) +$

$H(B_{i_2}) \geq H(S_{i(1-j)}) + H(S_{(1-i)j}) + H(S_{ij})$. This matches (within 1 bit) the upper bound given by Constr. 8.

Case 2: such i, j don't exist. Then the only non-trivial gates are XOR and $\neg$ XOR. GESS of Constr. 10, presented here for completeness, implements XOR and matches the lower bound of $H(S_{i(1-j)}) + H(S_{(1-i)j})$ for this case.

Construction 10. *(GESS ensemble for XOR gates.) Let $SEC = \{0,1\}^n$ and $TSEC = SEC^2$ be the secrets domains. Let the secrets tuple $\langle s_0, s_1 \rangle \in TSEC$ be given. The domains of shares are set as follows: $SH_1 = SH_2 = SEC$.*

Shr chooses $R \in_R SEC$ and sets $sh_{10} = R, sh_{11} = s_0 \oplus s_1 \oplus R, sh_{20} = s_0 \oplus R, sh_{21} = s_1 \oplus R$.

Rec proceeds as follows. On input sh_1, sh_2, Rec outputs $sh_1 \oplus sh_2$.

Theorem 10. *For each $n \in \mathbb{N}$, Constr. 10 is a GESS as defined by Def. 15.*

The proof of Thm. 10 is very simple and is omitted.

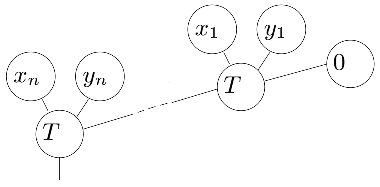
In conclusion, for the shares A_i and B_j of the two input wires, we proved

Theorem 11. *For every GESS scheme implementing an OR or an AND gate, when all secrets are chosen at random from the same domain SEC and each has entropy H_S, $\forall i, j \in \{0,1\} : \; H(A_i) + H(B_j) \geq 3H_S$.*

Of course, the entropy of each share must be at least H_S. Then all possible gates with two binary inputs are (almost) optimally implemented by either Constr. 8 or 10. Our Constr. 9 beats the lower bound of Theorem 11 by exploiting common information among secrets. We leave open the question of exact lower bounds for this interesting case. We stress that the share-size-to-secret-size ratio approaching 1, achieved by Constr. 9, is “near optimal”, since no construction can achieve ratio smaller than 1.

5.2.6 Application of GESS: Efficient Practical Two Millionaires

We apply the GESS approach to give a new efficient solution to the two millionaires problem. We design a GESS scheme for a new type of gate and use it to compute the Greater Than (GT) predicate. We use the intuitive circuit C (below) that compares bits of the parties' inputs x and y, starting with the most significant, and sets the answer bit when it encounters the difference.

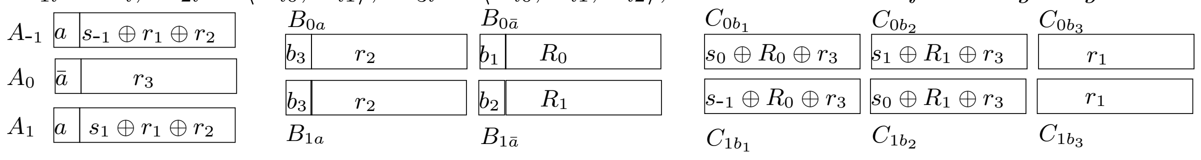


$$\text{where } T(j, x_i, y_i) = \begin{cases} j, \text{if} j \in \{-1, 1\}, \\ -1, \text{if} j = 0 \wedge x_i < y_i, \\ 0, \text{if} j = 0 \wedge x_i = y_i, \\ 1, \text{if} j = 0 \wedge x_i > y_i. \end{cases}$$

Here j is ternary input and x_i and y_i are bits. It is easy to see that C indeed computes GT: once a ternary wire is set to -1 or 1, that value is propagated to the output wire. We aim to minimize the expansion of the share corresponding to the input j. Note the double application of permute and point in Constr. 11. Denote by T-gate the gate implementing the function T above.

Construction 11. *(GESS ensemble for T-gates.) Let $SEC = \{0,1\}^n$ and $TSEC = SEC^3$ be the secrets domains. Let the secrets tuple $\langle s_{-1}, s_0, s_1 \rangle \in TSEC$ is given. The domains of shares are set as follows: $SH_1 = \{0,1\} \times SEC$, $SH_2 = (\{0,1\}^2 \times SEC)^2$ and $SH_3 = SEC^3$.*

Shr chooses $R_0, R_1, r_1, r_2, r_3 \in_R SEC$, $a \in_R \{0,1\}$ and $b = \{b_1, b_2, b_3\}$ - a random permutation of $\{0,1,2\}$, where each b_i is suitably represented by 2 bits. Shr sets the shares $sh_{1i} = A_i, sh_{2i} = \langle B_{i0}, B_{i1} \rangle, sh_{3i} = \langle C_{i0}, C_{i1}, C_{i2} \rangle$, as shown on the following diagram.



A_{-1} | a | $s_{-1} \oplus r_1 \oplus r_2$
A_0 | $\bar{a}$ | r_3
A_1 | a | $s_1 \oplus r_1 \oplus r_2$
B_{0a}: b_3 | r_2
B_{1a}: b_3 | r_2
$B_{0\bar{a}}$: b_1 | R_0
$B_{1\bar{a}}$: b_2 | R_1
C_{0b_1}: $s_0 \oplus R_0 \oplus r_3$
C_{1b_1}: $s_{-1} \oplus R_0 \oplus r_3$
C_{0b_2}: $s_1 \oplus R_1 \oplus r_3$
C_{1b_2}: $s_0 \oplus R_1 \oplus r_3$
C_{0b_3}: r_1
C_{1b_3}: r_1

Rec, on input $Sh_1 = a'r$, $Sh_2 = p_0 b_0\, p_1 b_1$, $Sh_3 = c_0 c_1 c_2$, outputs $r \oplus b_{a'} \oplus c_{p_{a'}}$.

Theorem 12. *For each $n \in \mathbb{N}$, Constr. 11 is a GESS as defined by Def. 15.*

Proof. Correctness of the scheme is easily verified. The simulator $Sim(s)$ chooses random $\alpha \in_R \{0,1\}, r'_0, ..., r'_4 \in_R SEC, \beta_0, \beta_1 \in_R \{0,1,2\}$, where $\beta_0 \neq \beta_1$. Let β'_i be suitable 2-bit representations of β_i. Sim outputs shares $\langle(\alpha r'_2)$,
$(\beta'_0 r'_0 \beta'_1 r'_1), (\gamma_0\gamma_1\gamma_2)\rangle$, where $\gamma_{\beta_\alpha} = s \oplus r'_2 \oplus r'_\alpha$, and the other two γ_i are assigned r'_3 and r'_4. The proof of equality of the generated distribution to the real execution is similar to that of previous two theorems, and is omitted. □

Performance. Let n be the length in bits of the compared numbers. The secrets corresponding to the T-gate at level i are of length i, and thus the secrets corresponding to the corresponding x_i and y_i are of lengths $3i$ and $2i+4$. Thus, Bob needs to send $\sum_{i=1..n} 3i = 1.5n(n+1)$ bits and perform n 1-out of-2 OT's with secrets of sizes $2+4, ..., 2n+4$.

The asymptotic complexity of this GT solution is worse than that of the best currently known for either setting with limited Alice (Yao's approach, see, e.g. [76]) or unlimited Alice ([41] or [14]). Still, our solution performs better for comparing smaller numbers ($n \approx 60..70$), since we do not use encryption[3].

We note that a reduction with a complexity similar to ours (quadratic) can be obtained by using BP-based techniques of [61].

5.3 Extension to Evaluating Polysize Circuits

When Alice is assumed to be polynomially bounded, all polytime computable functions can be efficiently evaluated. Beaver, Micali and Rogaway [5, 85], Naor, Pinkas and Sumner [76] and Lindell and Pinkas [70] suggested one-round protocols following Yao's [90] garbled circuit approach.

[3]This advantage is minute with standard (public-key primitive based) OT implementations; it may be significant in other settings.

As discussed, the OT reduction does not allow polytime evaluation of general polysize circuits, due to the exponential growth of combined secrets size for each level of general circuits. We now informally describe a natural extension that handles this problem in the standard model. This demonstrates the generality and applicability of the GESS approach. The resulting solution is essentially the standard variant of Yao's garbles circuit protocol. It is conceptually very clean, although slightly less efficient than the best known approach.

The protocol is essentially Constr. 7, with the following amendment. Bob will not propagate the secrets "up the circuit". Instead, for a gate G with output wires $w_1, ..., w_n$ and their (already computed) corresponding secrets tuples $(s_0^1, s_1^1), ..., (s_0^n, s_1^n)$, he encrypts all the secrets corresponding to each gate value *together*. More formally, he chooses two random keys k', k'' of a semantically secure private-key encryption scheme E. He computes $e_0 = E_{k'}(\langle s_0^1, ..., s_0^n \rangle), e_1 = E_{k''}(\langle s_1^1, ..., s_1^n \rangle)$ and assigns G's labels to be a random permutation of e_0, e_1. He then treats the keys as the secrets to be propagated, letting k' and k'' correspond to wire values 0 and 1 respectively. When Bob is done, he will have assigned secrets to each of the input wires and associated labels with each of the gates. He sends the secrets to Alice as before, additionally sending her the gate labels.

Alice obtains the secret shares for the input wires and proceeds evaluation similarly to the previous solution. The difference now is that, after having recovered a gate's secret (which is the key for one of the associated encryptions), she decrypts the corresponding encryption to recover the outgoing wires' secrets. To ensure that only one decryption succeeds, we impose an additional requirement on the encryption scheme. Informally, we need the ranges of encryptions under different keys be distinct, and that Alice is able to tell which decryption succeeded. This is a rather weak requirement, satisfied, for example, by schemes with *elusive* and *efficiently verifiable* ranges, formalized in [70]. Alice then uses the recovered secrets as shares in computing the child gate's secrets, and so on. Finally, she outputs the value of the output wire.

Theorem 13. *The above construction securely (against computationally unlimited Bob and limited Alice) reduces SFE of polysize circuits to Oblivious Transfer, in the semi-honest model.*

The proof of the theorem is rather intuitive and is delayed until Sect. 5.4.4.

The performance of the resulting approach is very similar to that of the currently best known solutions (e.g. [70, 76]). Indeed, our wire secrets are of the same size as theirs, and thus the only difference in performance is caused by the size of the gate labels. In [76], each gate has four labels of size N each[4], where N is the security parameter. It is easy to see that each gate of our solution adds up to $6N$ bits to the collection of all gate labels (two secrets of length N expand into two shares of length $N+1$ and two shares of length $2N$, which then are encrypted and stored as labels.). Some optimization of this number is also possible. For example, we need not encrypt (and thus add the corresponding labels) for the secrets that are just larger than N. This can reduce the gate induced label size gate by up to $2N$ bits.

We further note that in our scheme we only need to use encryptions once the secret sizes grow too large (i.e some threshold larger than encryption keys). Thus our method improves the performance of the evaluation of "the bottom part" of every circuit, and can be combined with Yao's garbled circuit implementations.

5.4 Formal Definitions and Proofs

In the previous discussion of this chapter, we presented special cases of definitions and intuition of proofs, which was sufficient to have a formal discussion, and, at the same time, allowed to concentrate on conveying the ideas of our constructions. In this section we present formal general definitions and proofs of theorems. The material of this section

[4]The authors also mention an optimization that allows using only three labels.

is a simple (but technical) formalization and generalization of the preceding material.

5.4.1 The General Definition of GESS

We give a general definition of a GESS scheme that allows to share a tuple of secrets. Let G be a gate with k inputs from domain $D_I = D_{I_1} \times ... \times D_{I_k}$ and one output from domain D_O. We also denote by $G : D_I \mapsto D_O$ the function computed by gate G. Let SEC be the domain of secrets and $TSEC \subset SEC^{|D_O|}$ be the domain of tuples of secrets to be shared. For simplicity of presentation and without loss of generality, assume that all domains D_{I_i} and D_O are initial sequences of non-negative numbers, e.g. $D_{I_1} = \{0, 1, 2, ..., |D_{I_1}| - 1\}$.

Definition 16. *(Gate evaluation Secret Sharing) A* gate evaluation *secret sharing scheme (*GESS*) for evaluating G (we also say GESS* implementing G*) is a pair of algorithms (Shr, Rec) (with implicitly defined secrets domain SEC, secrets tuples domain $TSEC$, k share domains $SH_1, ..., SH_k$ and k share tuples domains $TSH_1, ..., TSH_k$), such that the following holds.*

The probabilistic share generation algorithm Shr takes as input a $d_O = |D_O|$-tuple of secrets $\langle s_0, ..., s_{d_O-1}\rangle \in TSEC$ and outputs a sequence (of length of k) of tuples *of shares, where the i-th tuple $t_i \in TSH_i$ consists of $|D_{I_i}|$ shares $sh_{ij} \in SH_i$. The deterministic share reconstruction algorithm Rec takes as input a sequence of k elements $sh_i \in SH_i$, one from each domain, and outputs $s \in SEC$.*

Let $b = \langle b_1, ..., b_k\rangle \in D_I$ be a selection vector. Define the selection function
$Sel(\langle sh_{10}, ..., sh_{1|D_{I_1}|-1}\rangle, ..., \langle sh_{k0}, ..., sh_{k|D_{I_k}|-1}\rangle, b) = \{sh_{1b_1}, ..., sh_{kb_k}\}$.

Shr and Rec satisfy the following conditions:

- *correctness: for all random inputs of Shr and secrets tuples $\langle s_0, ..., s_{d_O-1}\rangle \in TSEC$, $\forall b \in D_I, Rec(Sel(Shr(\langle s_0, ..., s_{d_O-1}\rangle), b)) = s_{G(b)}$*

- *privacy (selected shares contain no information other than the value $s_{G(b)}$): There exists a simulator Sim, such that $\forall \langle s_0, ..., s_{d_O-1}\rangle \in TSEC$ and any $b \in D_I$:*

$$Sim(s_{G(b)}) \equiv Sel(Shr(\langle s_0, ..., s_{d_O-1} \rangle), b)$$

5.4.2 Proof of Theorem 7

Proof. Security against Bob is trivial since he does not receive any messages. The intuition for the scheme's security against Alice is that none of the GESS implementations leak any information. To prove security, we show how to construct Sim_A, perfectly simulating the following ensemble (view of Alice): $\mathrm{VIEW}_A(x, a) = \{x, m_{OT}, m\}$, where x and a are Alice's input and output, m_{OT} is the sequence of messages received from the OT oracles and m is the message received from Bob directly.

Sim_A first simulates wire secrets assignment as follows. He starts with the output wire, assigns its value to be a, and proceeds through gates from the bottom up as follows. Given gate G, its $GESS_G$, simulator Sim_G, and G's output wire value v, Sim_A assigns values to G's input wires according to $Sim_G(v)$.

Eventually, Sim_A assigns secrets to all input wires of C. Sim_A outputs $\{x, m'_{OT}, m'\}$, where x is Alice's input, m'_{OT} and m' are (proper representations of) the sequences of C's input wires assignments corresponding to Alice and to Bob respectively.

It is intuitive that the proposed simulator perfectly simulates Alice's view. Indeed, the vector of inputs to C defines a value assignment to each wire of the circuit, which, in turn, defines a distribution on shares/secrets obtained (received or computed) by Alice for each wire. We prove that wire assignment of Sim_A perfectly simulates the obtained secret for each wire. It is clear that Sim_A perfectly assigns the secret corresponding to the output wire by setting it to the output of the computation he obtained as its input. Further, Sim_A assigns secrets to the input wires of the output gate G. These secrets are distributed identically to the secrets that Alice reconstructs for these wires, because of the perfect simulation of Sim_G. Proceeding upward to the input wires, it is clear that Sim_A perfectly simulates all the wire assignments that Alice sees and reconstructs in the real execution. $\square$

5.4.3 The General Construction of GESS for AND/OR Gates

Construction 12. *(Improved GESS for gates with two binary inputs.) Let $D = \{0,1\}^k$ and $SEC = D^n$. Let secrets $s_{00}, ..., s_{11} \in SEC$ consist of n blocks of length k, and differ only in the j-th block. That is, let*

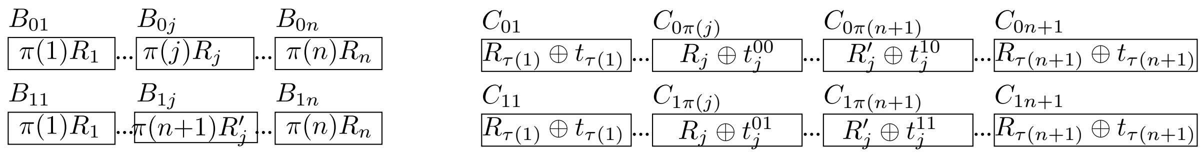
$$s_{00} = (\quad t_1 \quad \ldots \quad t_{j-1} \quad t_j^{00} \quad t_{j+1} \quad \ldots \quad t_n \quad),$$

$$...$$

$$s_{11} = (\quad t_1 \quad \ldots \quad t_{j-1} \quad t_j^{11} \quad t_{j+1} \quad \ldots \quad t_n \quad),$$

where $\forall i = 1..n$: $t_i, t_j^{00}, t_j^{01}, t_j^{10}, t_j^{11} \in D$, and the index j is determined only by the secrets. Let $TSEC \subset SEC^4$ be the space of all tuples of the above form.

Shr chooses $R_1, ...R_n, R'_j \in_R D$ and a random permutation[5] $\pi : \{1..n+1\} \mapsto \{1..n+1\}$. Let $\tau = \pi^{-1}$ be the inverse of π. For $m \in \{0,1\}$, Shr sets the shares $sh_{1m} = \langle B_{m1}, ..., B_{mn} \rangle$ and $sh_{2m} = \langle C_{m1}, ..., C_{mn+1} \rangle$, as shown on the following diagram.


B_{01} B_{0j} B_{0n}
$\pi(1)R_1$ ... $\pi(j)R_j$ ... $\pi(n)R_n$
B_{11} B_{1j} B_{1n}
$\pi(1)R_1$ ... $\pi(n{+}1)R'_j$ ... $\pi(n)R_n$
C_{01} $C_{0\pi(j)}$ $C_{0\pi(n+1)}$ C_{0n+1}
$R_{\tau(1)} \oplus t_{\tau(1)}$ ... $R_j \oplus t_j^{00}$ ... $R'_j \oplus t_j^{10}$ ... $R_{\tau(n+1)} \oplus t_{\tau(n+1)}$
C_{11} $C_{1\pi(j)}$ $C_{1\pi(n+1)}$ C_{1n+1}
$R_{\tau(1)} \oplus t_{\tau(1)}$ ... $R_j \oplus t_j^{01}$ ... $R'_j \oplus t_j^{11}$ ... $R_{\tau(n+1)} \oplus t_{\tau(n+1)}$

More specifically, the blocks of both shares of the first vector will be assigned $R_1, \ldots, R_n$, with the exception of the j^{th} block of the share corresponding to 1, which will be assigned R'_j. Shr then, for all i, prepends $\pi(i)$ to the i^{th} block of both shares of the first vector, with the exception of the j^{th} block of the second share, which gets prepended $\pi(n+1)$.

Each $\pi(i)$-th block of both shares of the second pair will be set to $R_i \oplus t_i$, with the exception of blocks $\pi(j), \pi(n+1)$. Those blocks assignment is motivated by Construction 8. Specifically, we set the $\pi(j)$-th block of the share corresponding to 0 to $R_j \oplus t_j^{00}$ and that block of the other share – to $R_j \oplus t_j^{01}$. We set the $\pi(n+1)$-st block of the share corresponding to 0 to $R'_j \oplus t_j^{10}$ and that block of the other share – to $R_j \oplus t_j^{11}$. This

[5] This permutation specifies which block of the second tuple is XOR'ed with the i^{th} block of the first tuple to obtain the i^{th} block of the reconstructed secret.

completes the description of Shr.

Rec proceeds as follows. He obtains two shares $sh_1 = (ind_1, r_1, ..., ind_n, r_n)$ and $sh_2 = (a_1, ..., a_{n+1})$. He reconstructs the secret $s = (\sigma_1, ..., \sigma_n)$ by setting $\sigma_i = r_i \oplus a_{ind_i}$.

Theorem 14. *For each $n, k, j \in \mathbb{N}$, Construction 12 is a GESS scheme as defined by Def. 15. (Note that security and correctness hold w.r.t. $TSEC$.)*

Proof. To prove security, we construct a simulator $Sim(s)$. On input $s = \sigma_1, ..., \sigma_n$, $Sim(s)$ does the following. He chooses random $r'_1, ..., r'_{n+1} \in_R D$ and a random permutation $\rho : \{1..n+1\} \mapsto \{1..n+1\}$. He outputs the shares $sh_1 = (\rho(1)r'_1, \ldots, \rho(n)r'_n)$ and $sh_2 = (\sigma_{\rho^{-1}(1)} \oplus r'_{\rho^{-1}(1)}, \ldots, \sigma_{\rho^{-1}(n+1)} \oplus r'_{\rho^{-1}(n+1)})$

We now prove that Sim perfectly simulates the real-life generated shares. The first share is distributed identically to both of the real-life generated shares of the first vector. Indeed, each r_i is distributed identically to each R_i, R_j and R'_j and $\rho(1), ..., \rho(n)$ is distributed identically to $\pi(1), ..., \pi(n)$ and to $\pi(1), ..., \pi(j-1), \pi(n+1), \pi(j+1), ..., \pi(n)$, for any j.

As for the second share, all blocks (and their positions) are generated identically to the real execution, with the exception of blocks in positions $\rho(j)$ and $\rho(n+1)$. Proof of the equality of their distribution to the corresponding blocks of the real distribution closely follows that of Construction 8 and is omitted. □

5.4.4 Proof Sketch of Theorem 13

Proof. (Sketch): The reduction is trivially secure against Bob, since he does not receive any messages from Alice. To prove security against Alice, we will show how to simulate the input wires' secrets and gate labels that Bob sends to Alice, given the output of the computation. We present the proof for binary fan-in 2 circuits; a more general argument is readily obtained by natural generalization.

The simulator $Sim(x, b)$ proceeds as follows. First, it (perfectly) simulates the secret of the output wire by s.

Then, for each level of the circuit, starting from the bottom, for each gate G of the current level: given the (previously simulated) G's output wires' secrets $s_0, ..., s_{k-1}$, it simulates G's input wires' secrets and gate labels as follows. It chooses two random keys s', s'' from the key domain of the employed encryption scheme. Then it computes $e_0 = Enc_{s'}(\langle s_0, ..., s_{k-1}\rangle), e_1 = Enc_{s''}(\langle 0, ..., 0\rangle)$ and assigns G's labels to be a random permutation of e_0, e_1. Then Sim runs the the simulator $S_G(s')$ of the secret sharing scheme of G. The simulator $S_G(s')$ produces two shares (distributed identically to real execution), each of which is the simulation of the secret of the corresponding wire.

Sim runs the above procedure on C "from the bottom up", and eventually obtains the simulations of the input wires and gate labels, which he outputs, suitably formatted.

We note the true randomness of all encryption keys and the perfect simulations of secret sharing schemes. Intuitively, the only way for an adversary to distinguish the simulation from the real execution is by distinguishing the sets of non-decrypted gate labels. However, learning anything "substantial" that way would mean breaking the semantic security of the employed encryption scheme, which can be shown by a simple hybrid argument. $\square$

5.5 Conclusions

In this chapter, we gave a more efficient information-theoretic reduction of SFE of a boolean formula F to oblivious transfer, of complexity is $\approx \sum d_i^2$, where d_i is the depth of the i-th leaf of F. It is not known whether information-theoretic efficient reductions exist for polysize circuits. Known constructions incur exponential overhead due to the need of "unwrapping" the circuit in to a tree. In addition, previous constructions suffered from exponential (in depth) cost of evaluation of each gate. With this work, we reduce the latter cost to quadratic.

Chapter 6

Key Exchange with Passwords and Long Keys

6.1 Introduction

Summary of the contributions of the chapter. We propose a new model for key exchange (KE) based on a combination of different *types* of keys. In our setting, *servers* exchange keys with *clients*, who memorize short passwords and carry (stealable) storage cards containing long (cryptographic) keys. Our setting is a generalization of that of Halevi and Krawczyk [58] (HK), where clients have a password and the public key of the server.

We point out a subtle flaw in some instances of the protocols of HK and demonstrate a practical attack on them, resulting in a full password compromise. We give a definition of security of KE in our (and thus also in the HK) setting and discuss many related subtleties. We define and discuss protection against *denial of access* attacks, which is not possible in any of the previous KE models that use passwords. Finally, we give a simple and efficient protocol satisfying all our requirements.

6.1.1 Motivation of the Problem and the Setting

In this chapter, we part with the problem of SFE, and discuss certain aspects of protection against external intruders.

We consider the goal of enabling multiple independent secure conversations between pairs of parties over an insecure network. The most convenient and natural way to achieve this is to perform a *Key Exchange* (KE), that is to provide the parties with matching randomly chosen keys that can be used for securing a particular conversation. Of course, each player wants to communicate with a particular person, and even a powerful adversary *Adv* should not be able to match him up with a wrong partner. Therefore, players must possess some secret information with which they can authenticate themselves. The kind of information that is available to players determines the setting of KE. The simplest KE setting is when players have a shared random and secret string. KE is more complicated in the public key setting, where parties have public/private key pairs with the public keys securely published. The most difficult setting is the pure password setting, where parties only have a short (presumably memorizable) shared password. We note that pure password KE protocols, at least in the standard model, are currently rather complicated and inefficient, due to the complexity of the setting.

6.1.2 Our Setting

Consider the client-server setting where both long keys and short keys (passwords) are used for KE. Assume that the server's (e.g. bank's) keys are securely stored. We take advantage of the inherent *logistical* differences in how keys are stored by the client (password in memory, long key on a storage card), to achieve more robust security than what is possible by using either type of key alone. Indeed, possession of long keys allows strong security guarantees against an online attacker. However, long keys can not be memorized, and thus must be stored, perhaps on a convenient plastic storage card. This is the vulnerability of this solution – the card may be (relatively) easily stolen by a physical attacker. On the other hand, passwords may be memorized, need not be stored, and thus can not be stolen. However, the protection against an online attacker one can hope to achieve with passwords is rather weak – passwords can always be guessed with relatively high probability. The only (somewhat satisfactory) protection against guessing attacks is recognizing them and refusing connection after a predetermined number of password failures[1].

Combining the benefits of both settings allows us to obtain a system, secure against both *types* of attack, and thus suitable for protection of sensitive information. This model is even more appealing due to its wide acceptance – it is natural for us to think of a card and a password, when we do, say, personal banking. More motivation is given in Sect. 6.3.

6.1.3 Our Contributions

We demonstrate a dangerous practical attack on some instances of the Halevi and Krawczyk (HK) [58] protocols, resulting in full compromise of any client's password (Sect. 6.2). The elegance, simplicity and practicality of the HK model and protocols resulted in their widespread practical use (e.g. their variants are being considered for parts of the IETF key exchange standard [42, 28]). Therefore, the discovery of our attack may also have an important practical impact. We stress, however, that the standardized variants of HK protocols do not suffer from the discovered vulnerability.

We propose and advocate the above *Combined Key* model of key exchange (ckKE[2]). To the best of our knowledge, it has never been formally discussed. ckKE is a generalization of the HK model.

[1]We mention (but do not explicitly address) a variation of this defense against "too many" password guessing attacks. There the server limits the rate with which logins can be made, e.g. by exponentially increasing wait times between unsuccessful logins.

[2]We choose this capitalization to be consistent with the KE literature.

We give a formal definition of security of ckKE (Sect 6.3). Defining KE even in simpler settings has proven to be notoriously difficult, with a variety of seemingly innocuous decisions to be made. We discuss the subtleties of many of our choices, such as the necessity of tightness in the allowed success of the adversary, distinguishing the types of failures and reporting them, etc. Much of our discussion (e.g. on tightness of allowed success of the adversary *Adv*) also applies to and benefits pure password models.

We aim to make our definition as simple and natural as possible. For example, we require the server to explicitly indicate in its output whether a password failure occurred. We find this more intuitive than defining password guessing attack as an act of interference by the adversary (e.g. a successful impersonation!), as done in previous formalizations, such as [58, 10]. Moreover, in previous formalizations, such as [58, 10, 17], the attacks are accounted by the environment; the server may not even "know" they occurred (e.g. in case of successful impersonation), which makes attack recognition in practice less intuitive. We also find the game style of definitions (used in this work) generally simpler and less prone to error than the simulation style (see discussion on the style of definition in Sect. 6.3.1 for more details). We wish to stress, however, that we don't claim finding flaws in previous definitions, and, in particular, the discovered flaw of the Halevi-Krawczyk protocol is not inherent in their model.

We discuss unique security features available in ckKE "for free", such as the possibility of protection against the following *Denial of Access* (DoA) attack. *Adv*, attacking a player P, tries to connect to P's partner Q, using any password *pwd*. If *pwd* is correct, *Adv* wins; if not, *Adv* continues until he wins or Q refuses to connect to P. Then a legitimate P can no longer connect to Q. This easy to mount attack is unavoidable in any password-based setting (including HK) and is highly disruptive. We are not aware of the prevention of this attack being previously formalized. We formalize this attack and show how to prevent it in our model.

Finally, we give a simple and efficient *two flow* KE protocol and prove its security

(Sect. 6.4). An important feature of our protocol is that its flows are *independent* of each other, and thus can be sent in any order (or simultaneously), allowing for more flexibility and round efficiency.

6.1.4 Related Work

The problem of key exchange has deservedly received a vast amount of attention (e.g. [34, 11, 66, 7, 87, 25, 26]). The more complicated setting of pure password-based KE (pwKE) was first considered by Bellovin and Merritt [12]. Formal definitions (and protocols) in this setting were given by Bellare, Pointcheval and Rogaway [10], Boyko, Mackenzie and Patel [18], Goldreich and Lindell [51], and, recently, by Canetti et al. [24], as well as by many others.

Most relevant to our work is the problem of password-based KE in the asymmetric client-server setting, where the client has a password and the public key of the server. The question of resistance to off-line password-guessing attacks in this setting was first raised by Gong, et al. [55]. Later, Halevi and Krawczyk [58] formalized the notion of *one-way password authentication* in this setting and gave simple and efficient protocols realizing it. They also extended their protocols to achieve key exchange with mutual authentication and perfect forward secrecy. The HK model is much simpler than the pure password model. The work of HK was the inspiration of our work.

Further, Boyarsky [17] criticised the protocols of the earlier version [57] of [58] and suggested his own formalization of the same model. He showed several ways to amend a variant of protocols of [57] to satisfy his definition. We stress that he does not criticize protocols of the later version [58] we are considering.

Pinkas and Sander [80] consider heuristic approaches to securing password-only based authentication. They increase the cost of password-guessing and DoA attacks by using reverse Turing tests (RTT), that is, problems that are easy to solve for humans, but not for computers. We approach a different problem. In particular, RTT techniques can not increase security of a particular client against a determined attacker.

6.2 Attacking the Protocols of Halevi and Krawczyk [58]

Halevi and Krawczyk give four versions of their protocol (suitable for different tasks: password transmission, one-way authentication, and key exchange in two settings). Three of the four versions (with the exception of the Encrypted Password Transmission protocol) are (similarly) affected. We demonstrate our attack on their key exchange protocol.

The Halevi-Krawczyk protocol. Let S be a server with the public key pk_S, and p be the password shared between S and the client C. Let function $f(\cdot;\cdot)$ be *one-to-one on its components*, i.e. for every fixed strings p, x, functions $f(p;\cdot)$ and $f(\cdot;x)$ are one-to-one. Let $E = (Gen, Enc, Dec)$ be a CCA secure encryption scheme (See Sect. 2.7.1 for definitions of encryption schemes and CCA security).

Construction 13. *(The Halevi-Krawczyk Mutual Authentication and Key Exchange Protocol (Π_{HK}))*

S		C
pick a nonce n	$n, pk_S \rightarrow$	*verify* pk_S
		pick random long key k
	$\leftarrow C, n, Enc_{pk_S}(k, f(p; C, S, k, n))$	
decrypt and verify		
$y := PRFG_k(n, S, C)$	$y \rightarrow$	*check* $y = PRFG_k(n, S, C)$
set $K = PRF_k(y)$		*set* $K = PRF_k(y)$

The "decrypt and verify" step outputs "FAIL" if the encryption is invalid or the received value of f does not match what S computes himself. The *nonces* must satisfy the *only* requirement that they never repeat.

Our Attack. We exploit the structure of f. Although Halevi and Krawczyk only require that f is one-to-one on its components, their proof actually assumes that $f(\cdot; C, \cdot) \neq f(\cdot; C', \cdot)$ for any unequal client names C, C', as evident from Footnote 9 on p. 258 of [58]. We will refer to this assumption as the *HK-New* requirement. We show that the HK-New requirement is essential – use of some functions f, satisfying the HK, but not the HK-New, requirement, allows for password guessing attacks. We note that it is possible to make the proof (of security of one-way password authentication protocol) of Halevi and Krawczyk protocol go through by adding the HK-new requirement.

For simplicity, we describe our attack on a specific instantiation of Π_{HK}. We stress that natural variants of our attack apply to many choices for f that do not meet the HK-New requirement, for many nonce strategies, as well as for other parameter settings.

Let client names and passwords be 10 bits long, and nonces be 30 bits long. For a variable V, let v_i be the i-th bit of V. For example, $C = \langle c_1, c_2, ..., c_{10} \rangle$ is the name of the honest player, and $n = \langle n_1, n_2, ..., n_{30} \rangle$ is the nonce. Let the function be $f(p; C, S, k, n) = \langle c_1, ...c_9, c_{10} \oplus p_1, n_1, ...n_{21}, n_{22} \oplus p_2, ..., n_{30} \oplus p_{10}, S, k \rangle$. Finally, let nonces be chosen sequentially starting from 0. Note that this is a valid configuration of Π_{HK}.

The attack proceeds as follows. *Adv* interacts with an honest server S, and attacks an honest client C with any name $C = \langle c_1, c_2, ..., c_{10} \rangle$. *Adv* registers with S a bad client B with the name $B = \langle c_1, c_2, ..., c_{10} \oplus 1 \rangle$ and a randomly chosen password $p' = \langle p'_1, ..., p'_{10} \rangle$. Let p be C's password. Suppose for now that $p_1 \neq p'_1$, i.e. passwords of C and B differ in the high order bit. *Adv* observes one execution of KE between S and C. *Adv* records the encryption e sent by C and the nonce n (for concreteness, say $n = 00..00$, e.g. n is the first nonce). Now, B logs into S as himself, as follows. S sends the nonce $n' = n + 1 = 00..01$, and B replies with $\langle B, n', e \rangle$. Now, if S doesn't fail, the password of C is computed as $pwd = \langle p'_1 \oplus 1, n_{22} \oplus n'_{22} \oplus p'_2, ..., n_{30} \oplus n'_{30} \oplus p'_{10} \rangle$ (since for $i = 22, ..., 30$, it must be that $n_{i-20} \oplus p_i = n'_{i-20} \oplus p'_i$). Also, if $p = pwd$, then S must accept, since

$f(p'; B, S, k, n') = f(pwd, C, S, k, n)$. Thus, if S fails, pwd is eliminated from the possible passwords list.

B proceeds logging in as himself another $2^9 - 2$ times, eliminating different passwords one by one, until S accepts and that fact determines C's password. If S does not accept after B logged in $2^9 - 1$ times, B changes the first bit of his password with the server, and repeats the above entire attack (say, starting with a nonce ending with nine zeros), searching the other half-space. Finally, the two possible unchecked passwords can be verified by the same approach (and changing the password of B).

We stress that there were no attempts at impersonating C or S, and *all* failures are attributed to B. Neither C nor S know that C was attacked, thus C's account is never blocked. If B's account is blocked due to failures, B can claim mistyping and restore access. Moreover, there is no need to attack from only B's account; the attack can be easily distributed to try only a few passwords from each of many bad accounts. Again, it is easy to see that our attack is naturally generalizable to many practical instantiations of Π_{HK}.

On Boyarsky's [17] amendments of HK. The earlier version [57] of [58] had essentially the same protocol as [58], with the exception of the imposed requirements on the encryption scheme ([57] only required so-called *one-ciphertext verification attack* resistance, vs *ciphertext verification attack* resistance in [58]). Boyarsky [17] (independently from the revision resulting in the current version [58]) discovered the insufficiency of the weaker encryption. He gives his own formalization of the model and suggests three different amendments (see Sect. 5 of [17]) of the protocols of [57]. Boyarsky limits his consideration to the case where f is a concatenation function; thus our attack is not applicable to his protocols.

6.3 Key Exchange in the Combined Keys Model

Recall from the discussion in Sect. 6.1 that our setting (client carrying a plastic storage card and remembering a password) allows the advantage of robustness, that is graceful degradation of security in case one of the two types of keys is compromised. In particular, if the client's password is compromised, the security of KE should not suffer. On the other hand, if the card is compromised (e.g. copied), the remaining security should be that of the HK password model.

On resistance to server compromise. Halevi and Krawczyk briefly discuss resistance to insider attacks, i.e. attacks by rogue server employees who have access to some, but not all, private information stored on the server (see Sect. 3.3 in [58] for discussion of heuristic defense approaches). As another advantage of our setting, we mention that it allows stronger protection against server compromise. For example, since our clients have storage cards, public/private key pairs for each client C_i can be set up and used appropriately. Of course, an attacker who steals all the server data would now be able to successfully pose as the server. However, he can be prevented from posing as a client, as long as the client's private key remains secret. We note that such protection will require significant additional complexity of the definition and the protocol, and we leave it outside the scope of this work. Therefore, for clarity, as do Halevi and Krawczyk, in our main exposition we assume that the server's private information is never compromised. For completeness, after presenting our protocols in Sect. 6.4, we briefly discuss how to modify them to protect against some consequences of server compromise.

On Denial of Access (DoA) attacks resistance. Recall that in the HK (and also in the pure password) setting, security critically depends on the ability of servers to suspend clients' accounts if there are "too many" password failures. At the same time, it is all too easy for *Adv* to cause them, making systems unusable by a trivial and easily mounted attack. In our combined key setting, it is natural to introduce protection against such DoA attacks. This can be done by requiring that polytime attackers can not cause password failures (and thus account suspension) without possession of long keys, stored on the card of the client. Of course, *Adv* may attempt attacks even without having the long keys, and furthermore, such attacks may be noticed by the servers. However, it is not hard to ensure that *Adv* does not learn anything from such attacks. This can be done, for example, by server first verifying possession of the long key (e.g. in form of a MAC), and immediately failing, if such verification failed. Then *Adv* does not learn anything about *pwd*, since it was not even used by the server. Therefore, such password guessing attacks are not a threat, and can be ignored. We formalize resistance to DoA attacks in our definition.

In our view, the main reason for using two types of keys is the two qualitatively different layers of protection against compromise. DoA resistance, although an important bonus, may not alone justify the cost of long key storage and management.

The reader may ask why one can't simply do two KE's in the two relevant models (one with parties sharing long keys, and the HK model) and combine the keys to obtain a KE protocol in our model. There are a number of issues to be addressed there. Firstly, a definition of security has to be given anyway – which is the bulk of our work. We note that some of the definitional subtleties arise specifically due to the use of both keys simultaneously. These subtleties cannot be addressed in either of the simpler models separately. See, e.g., the discussion on password updates in Sect. 6.4. Secondly, natural ways of combining the two KE protocols (such as establishing a secure session using long keys, and sending the password over it) result in less efficient protocols.

6.3.1 Pre-definition Discussion

We start by briefly recalling the general setting for KE. There is a number of players (in our case, they are divided into two types – clients and servers) who have associated credentials, and pairs of whom may have shared common information. We think of a player as an *identity*, which may have many *instantiations*. Whenever a player P wishes to talk to another player Q, an instance of P is created with the required credentials passed. Thus an instance can be thought of as a participant of a particular conversation.

It is convenient to separate the notions of identity and instance for several reasons. Firstly, it is easier to talk about the independence of instances. Independence is highly desirable to avoid maintaining state and worry about communication and synchronization between instances. Secondly, a need often arises to have several channels of communication open between two or more parties simultaneously. Then the notion of instance makes it easier to implement and model concurrent executions of KE by a player.

We do not discuss how a player P knows that he wants to talk to a player Q. This may be done as part of previous (possibly insecure) communication, scheduled to happen at some predetermined time, or be requested by a higher level protocol. We give *Adv* the power to initiate conversations between players to model all possible scenarios.

Our goal is to enable a secure conversation, or *session*, between the instances of two players. Key exchange provides corresponding pairs of participants with matching keys that can be used for securing their communication. Of course, the keys of honest parties must appear random to the adversary *Adv*, and *Adv* must not be able to cause instances to match up in an inconsistent way[3].

To formalize the latter requirement, we need to define the notion of *partners* – instances who end up having an intended conversation. We use session IDs (SID) to partner instances of players. There are several ways of using SID for this purpose, and we choose what we find to be the most natural – requiring each party that output a key to have an additional output *sid*. The other ways (e.g. requiring *sid* to be an input to parties, or requiring existence of a partnering function) seem to be less intuitive. We note that many natural protocols can be naturally modified to produce session ids. The *sid* output is not necessary in real protocols; it is only used for the purpose of defining and analyzing

[3]We note that *Adv* can cause confusion by mismatching instances of players and making them output *unrelated* keys. We don't regard this as a problem.

security of KE protocols.

Definition 17. *(KE Partners) Let P be a player. We denote by P_i the i-th instance of P. We write P_i^Q to emphasize that P_i intends to do KE with (some instance of) player Q. We say that an instance C_i^S of a client C and an instance S_j^C of a server S are* partners*, if they have output the same session id sid.*

Note that no two instances are partners when they are created; they may become partners once they've executed their KE protocols. We stress that P_i and P_i^Q refer to the *same* instance of P. We may omit the superscript in P_i^Q, when it is clear from the context.

Mutual authentication (MA). MA is an assurance that, if P_i^Q successfully completed and output a key, there must have been a Q_j^P "communicating" with him. We choose not to require it, because it can be achieved at the cost of two additional "key confirmation" flows (and refreshing the session key). Moreover, P_i^Q can never be sure that Q_j^P "is there" anyway, since Q_j^P may go offline at any time. Note, it is rather common and accepted to not require explicit mutual authentication for these reasons (e.g. [24]). Further, if we required MA, we must use a special $\perp$ output symbol to denote failure. In our definition we allow $\perp$, but don't insist on its use.

On the notions of attacks and failures. We first note that a special kind of failure – the *password failure* – must be introduced in our model to allow protection against DoA attacks. Intuitively, if *Adv*'s attack is such that the act of failure of the server may reveal some information about the client's password, then such failure is a password failure.

A natural approach to define adversary's ability to attack the system is by counting password checking *attempts*. However, it is less natural to define what an "attempt" is. Indeed, previous works on password-based key exchange (e.g. [58, 10]) define "attempt" essentially as the act of *Adv*'s interference with the exchange of messages between two parties. However, it is less clear, for example, whether an act of *Adv* changing an insignificant bit of a message or an act of successful impersonation is such an attempt.

Moreover, previously, the number of attempts was counted not by the server instances (they are not required to "know" whether a password guessing attack occurred), but by the environment.

An important feature of our definition is that servers themselves determine when, whether and what type of failure occurred. This explicates the notion of a *failed password attempt*, and ensures server's ability to identify a threat and react to it. Therefore, depending on the kind of failure, we allow servers to output either a *failure* symbol $\perp$, or a *password failure* symbol P$\perp$. We count password failures as P$\perp$'s reported by the servers, and clients accounts are suspended (to prevent further password guessing) based solely on that information and a predetermined threshold q. Therefore, a misidentification of an attack by the server is an omission of the protocol (opening a possibility of either password checking or DoA attacks), and we deem such protocols insecure.

We note that previous definitions, such as those of [58, 10, 17], can be similarly amended to ensure "explicit authentication" by additionally requiring that the server output P$\perp$ when he thinks a password attack has occurred. However, as discussed above, it seems to be cleaner to use the server's output as the only criterion for determining whether such an attack took place. Further, to ensure that the server does not misidentify the attacks, his output would need to be incorporated into the definitions, further complicating them.

On the style of definition. As mentioned earlier, we prefer the game style of KE definitions in this work. We find it easier to understand, since the game of the definition naturally corresponds to the actions and abilities of the adversary. We don't seem to need the complexity of simulation style definitions. An exception seems to be the complex universally composable (UC) definitions, which can model subtle issues such as password mistyping (see [24] and discussion in Sect 6.3.3). In addition to their complexity, UC-secure protocols currently are significantly less efficient than protocols in other frameworks. From another point of view, it is highly desirable to have different styles of definitions to discuss their relative strengths and, hopefully, prove equivalence in some settings. This work would help us gain confidence in the KE definitions we eventually converge on.

On modelling the adversary. We consider a powerful *Adv*, who schedules events (such as creation of players and their instances) and controls all communications. This latter is modelled by the parties not sending messages to each other, but giving them to *Adv* for delivery. *Adv* is allowed to arbitrarily modify the messages (including dropping and injecting them) and schedule delivery. We allow *Adv* to create and arbitrarily initialize a polynomial number of accounts for corrupted clients. Note that in this model the actions of corrupt players need not be discussed separately from the actions of *Adv*, since *Adv* can simulate all their actions. For example, a message sent by a corrupted party can be viewed as a message injected by *Adv*.

Recall, *Adv* steals either the long key or the password of a client, and attacks one of the several security features of the protocol. We describe the (five) possible settings as games the attacker plays. (These games cover all cases – the cases that are not discussed explicitly are implicitly covered by stronger settings.)

Game KE_1 models the most complicated setting where *Adv* stole the long key of the client, and is attacking a server (that is trying to distinguish server's session key from random). This is the only game where *Adv* can benefit from guessing a password. Thus, in KE_1 *Adv* is allowed a limited number of $\mathrm{P}\perp$'s.

Game KE_2 models the setting where *Adv* stole the long key and the password of the client, but is attacking a client.

Game KE_3 models the setting where *Adv* stole only the password of the client, and is attacking a server.

Game DOA models the inability of *Adv* to cause password failures without stealing the long key.

Game SID models the inability of *Adv* to cause two honest parties output different session keys, and is included for technical reasons (see discussion before the game's definition in Sect. 6.3.2 below).

One way to define security is to describe one adversary who, at some point in his attack, decides which of the five games above he really wants to play. However, since *Adv*'s breaking abilities vary significantly among the games, defining allowed success of *Adv* in a "combined" game would be unnecessarily complicated. Therefore, we choose to describe five adversaries, each playing the corresponding game. We define the security of ckKE by inability of any of adversaries to win any of these games "too often". We note that it is possible to define the "combined" adversary model carefully, and to prove that any protocol that is secure with respect to the five adversaries would also be secure with respect to one "combined" adversary.

Liveness. Note that protocols may never terminate (e.g. when *Adv* cuts the communication channels). Instances may also output special failure symbols instead of (*sid*, key) pairs (e.g. when they detect *Adv*'s interference). To ensure usability of KE protocols, we disallow these exceptional cases, unless *Adv* indeed attacks the system. Thus, we require that in the absence of an adversary, when processes communicate as intended, all sessions terminate, and intended partners output the same session id and key.

6.3.2 Formal Definition of Security of Key Exchange in the Combined Keys Model

Let n be a security parameter, and m be the number of bits in the password. In general, m can be a function of n; interesting cases are when m is constant or logarithmic in n. WLOG, say, the password domain is $D = \{0,1\}^m$. All players (*Adv*, clients and servers) are p.p.t. machines. Recall, the notion of partnering is defined in Def. 17.

We start by presenting the KE games. Recall, the first game models the setting where *Adv* obtained the long key of the client, is attacking a server, and is allowed a limited number of P⊥'s.

Game $\mathbf{KE}_1$. *The adversary Adv starts by deterministically choosing the active attack threshold $q \in 1..|D|$ (based on the security parameter n) and creating an (honest) server S. Adv chooses S's name; then S's public and private keys are set up, and only the public key revealed to Adv. Adv then runs the parties by executing steps 1-5 multiple times, in any order:*

1. *Adv creates an honest client C. Adv is allowed to pick any unused name for the client; the client C is registered with S, and long key ℓ and password pwd are set up and associated with C. Only one honest client can be created. Adv is given the long key ℓ, but not pwd.*

2. *Adv creates a corrupt client B^i. Adv is allowed to initialize him in any way, choosing any unused name, long key and password for him.*

3. *Adv creates an instance C_i of the honest client C. C_i is given (secretly from Adv) as input: his name C, the partner server's name S, the public key of S, the long key and the password of C.*

4. *Adv creates an instance S_j of the honest server S. S_j is given (secretly from Adv) as input: his name S, the private key of S, the partner client's name (C or B^i) and that client's long key and password.*

5. *Adv delivers a message m to an honest party instance. That instance immediately responds with a reply (by giving it to Adv) and/or terminates and outputs the result (either a (sid,session key) pair or the* failure symbol $\perp$*) according to the protocol. The server instance can additionally output the* password failure symbol $P\perp$*. If the total number of $P\perp$ for the honest client is equal to the threshold q, Adv becomes restricted – he can not deliver messages to any instances S_j^C.*

   Adv learns the output, with the exception of its session key part. Additionally, at any time Adv may "open" any completed honest instance – then Adv is given the *session key output by that instance.*

Then Adv asks for a challenge on an instance S_j^C of the server S. S_j^C, who has been instantiated to talk to the honest client C, must have completed and not failed. The challenge is, equiprobably, either the key output by S_j^C or a random string of the same length. Adv must not have opened S_j^C or a partner of S_j^C, and is not allowed to do it in the future.

Then Adv continues to run the game as before (execute steps 2-5). Finally, Adv outputs a single bit b which denotes Adv's guess at whether the challenge string was random. Adv wins if he makes a correct guess, and loses otherwise. Adv cannot "withdraw" from a challenge, and must produce his guess.

Note the following technicality of KE_1. It is possible that *Adv* may find himself unable to complete the game. This may happen only when he had just caused the q-th P$\perp$ (and hence he is not allowed to deliver messages to servers) and he has no completed instances whom he is allowed to challenge. One way to handle this would be to require *Adv* flip a coin to determine whether he won or lost. We prefer to simply disallow, by this discussion, such behaviour of *Adv*, since the stalemate can be easily avoided by *Adv* having a "safety instance" complete before he risks the q-th P$\perp$.

In all other KE games (KE_2, KE_3, SID and DOA) below, it is possible (and natural) to require that the knowledge of *pwd* does not help *Adv*. We thus choose to reveal the password to *Adv* and remove restrictions on the number of P$\perp$'s (thus removing the definition of q). These games are presented by modifying KE_1. All of the above three modifications are included in all games below (and the last two are omitted in individual descriptions for conciseness).

Game KE_2 models the setting where *Adv* stole the long key and the password of the client, but is attacking a client.

Game KE_2. *This game is identical to KE_1, with the following additional exceptions.*

- *Adv is given pwd (in addition to ℓ) and must challenge an honest client instance C_i^S, who is talking to S.*

Game KE$_3$ models the setting where *Adv* stole only the password of the client, and is attacking a server.

Game KE$_3$. *This game is identical to KE$_1$, with the following additional exceptions.*

- *Adv is given pwd, but not the long key ℓ.*

Game SID enforces a non-triviality condition, preventing parties from improperly partnering up (e.g. by unnecessarily outputting the same session ids). Recall, *Adv* is not allowed to challenge parties whose partner has been opened, and we need to ensure that *Adv* is not unfairly restricted.

Game SID. *This game is identical to KE$_1$, with the following additional exceptions.*

- *Adv is given pwd (in addition to ℓ) and does not ask for (nor answers) the challenge.*
- *Adv wins if any two honest partners output different session keys.*

Finally, game DOA models resistance to the Denial of Access (DoA) attacks.

Game DOA. *This game is identical to KE$_1$, with the following additional exceptions.*

- *Adv is given pwd, but not the long key ℓ.*
- *Adv does not ask for (nor answers) the challenge.*
- *Adv wins if a server instance S_j^C outputs $P\bot$.*

Definition 18. *(Secure Key Exchange in the Combined Keys Model.) We say that a key exchange protocol Π is* secure in the Combined Keys model*, if for every polytime adversaries Adv_1, Adv_2, Adv_3, Adv_{sid} and Adv_{doa} playing games KE$_1$, KE$_2$, KE$_3$, SID and DOA, their probabilities of winning (over the randomness used by the adversaries, all players and generation algorithms) is at most only negligibly (in n) better than:*

- $1/2 + \frac{q}{2|D|}$, *for* KE_1,
- $1/2$, *for* KE_2 *and* KE_3,
- 0, *for SID and DOA.*

KE definition for the HK setting. We note that Halevi and Krawczyk do not formally define the full notion of KE in their setting, but concentrate on the *one-way password authentication* of the client to the server. Because ckKE is a generalization of the HK setting and thanks to the modularity of our presentation, it is not hard to extract the KE definition for the HK setting from Def. 18. The only difference between our and the HK settings is that we additionally allow for the use of the long shared key ℓ. It turns out that it suffices to remove the games that do not allow *Adv* to know ℓ from Def. 18, to obtain a definition for the HK setting. (Of course, we also need to remove the uses of the long key ℓ from the remaining games.) Indeed, it is not hard to verify that the remaining games cover all possible attacks *Adv* can do in the HK setting. We explicate this definition below.

Definition 19. *(Secure Key Exchange in the HK Model.) We say that a key exchange protocol* Π *is* secure in the Halevi-Krawczyk, or *hybrid*, model*, if for every polytime adversaries* Adv_1, Adv_2 *and* Adv_{sid} *playing (amended as described above) games* KE_1, KE_2 *and SID, their probabilities of winning (over the randomness used by the adversaries, all players and generation algorithms) is at most only negligibly (in* n*) better than:*

- $1/2 + \frac{q}{2|D|}$, *for* KE_1,
- $1/2$, *for* KE_2,
- 0, *for SID.*

We note that although the pre- and post-definition discussion (of Sect. 6.3.1 and 6.3.3) discusses the ckKE setting, much of it applies to the HK setting as well.

6.3.3 Post-definition Discussion

On the sufficiency of only one honest server and one honest client. We note that definition of security is not strengthened by allowing *Adv* to create additional (good or bad) servers or good clients. The reason for this is that we assume independence in the initialization procedures of each pair of identities, and each instance is initialized only with information relevant to its partner. More detail follows.

Consider an adversary who wishes to attack a particular player – a client C or a server S. Suppose we allowed creation of additional good or bad servers. Note that initialization of a client C proceeds independently for servers S^1 and S^2, and, further, $C_{i_1}^{S^1}$ has no information about $C_{i_2}^{S^2}$, that is not known to *Adv*. Therefore, creating accounts for C with more than one server and instances of C talking to them does not help *Adv*, since it can be simulated by *Adv*. On the other hand, the ability to create many clients with a server is essential, since server instances talking to different clients do share common information among themselves – the secret key of the server. In fact, we exploit that in our attack on Π_{HK}. Only one honest client is sufficient, however, since additional honest clients can be played by *Adv*. We note that had we allowed clients to possess information common to two or more servers, we would have to allow *Adv* to create additional bad servers.

On the order of creation of good client and revealing the long key ℓ. *Adv* should first create the good client, and only then be allowed to see ℓ. This is the way the attack works in real life. Had we reversed the order, it would be easy to construct good protocols that would be defined insecure (e.g., a server leaks some information, if the client's name is the same as ℓ.)

On the allowed success of *Adv* in KE$_1$. Consider the success an adversary can always achieve (and therefore must be allowed in our definition). After q queries, *Adv* can guess the password with probability $q/|D|$, and if he fails to guess it, he can distinguish the key from random with probability $1/2$. Therefore, we should allow *Adv*'s probability of success of at least $\frac{q}{|D|} + \frac{1}{2}\frac{|D|-q}{|D|} = \frac{1}{2}\frac{q+|D|}{|D|} = \frac{1}{2} + \frac{q}{2|D|}$.

On independence of the states of instances. In our model, there is no global information, and state is not preserved between executions of instances of players. Therefore, for example, it is not possible for an instance to know exactly how many P⊥'s occurred. Nevertheless, some communication and preservation of state can be achieved with the help of the adversary, as follows. The private key of S now additionally includes an n-bit MAC key k_M. Whenever S_j wants to publish a message m, he gives $(m, MAC_{k_M}(m))$ to *Adv*. The server's protocol has an optional field in one of the expected messages. S_j only accepts the properly MAC'ed messages in that field (this is essential, so that *Adv* cannot forge messages). We stress that communication may only happen if it is in the interest of *Adv*. Therefore, it can not be used to increase security of protocols, but mainly to uncover weaknesses of definitions (see example in the next topic).

On continuing the game after q P⊥'s. In the real world, at least ideally, after q P⊥'s, the server knows there is an attack on C, and will not accept new connections and will terminate all incomplete instances. How should we model this in our KE games? Although S may have cut communication with C, old sessions may still exist, and we need to ensure that they remain secure. That is why we allow the game to continue as before, but disallow sending messages to the server instances after q P⊥'s occurred.

Observe that once *Adv* got the challenge, "trying" another password may not help him much. Therefore, in particular, it is crucial to allow to challenge instances *after* q P ⊥'s occurred.

It is not hard to design a concrete protocol demonstrating the necessity of our choice. Take a secure protocol Π. Modify it as follows to obtain Π'. Once a P⊥ of an honest client C occurred in the game (see above discussion on independence of states), in all future sessions with instances of C the all-zero session key is chosen with fixed small, but non-negligible probability (say $prob = \frac{1}{|D|^3}$). Clearly, this is a bad protocol, since after performing only one active attack, an attacker certainly breaks into one of the next few sessions. However, Π' would be deemed secure according to the definition, if *Adv* is not allowed to challenge after q P$\perp$'s (this is because *Adv* is allowed only one challenge, and he does not know which is the weak session. The expected advantage of *Adv* is less than what he gets from the q-th password try.)

On the necessity of tightness in defining the allowed success of *Adv*. Note that for every non-negligible slack allowed in *Adv*'s success, there is a natural variant of Π' above, deemed secure by such definition. While one may be tempted to not be very careful in denying *Adv* "a few extra password tries", Π' has a much more dangerous vulnerability, which really should be prevented. We remark that in the password-only setting, if an indistinguishability of challenge based security definition does not require tightness, a simpler variant of Π', where players *always* output an all zero key with sufficiently small (yet non-negligible) probability, would be deemed secure.

On clients mistyping the passwords. How should we model the case when an honest client mistypes the password and causes P$\perp$? Consider the following protocol. Take a secure protocol, and modify it, so that S_j^C reveals ℓ once P$\perp$ occurred. It is easy to see that the new protocol remains secure in our definition, since we implicitly assume that C never mistypes the password. Indeed, in our definition, if a P$\perp$ occurred, it must have been caused by *Adv*. Since *Adv* cannot cause P$\perp$ without possession of ℓ, it is OK if S_j^C reveals ℓ. However, intuitively, we would not want to call such a protocol secure.

The only way to formally address the issue in our model is to allow C to mistype the password. A natural first idea is to allow *Adv* to instantiate clients with the password of his choice. However, it is not clear that this models real life – most often clients mistype their passwords to something related.

A natural next idea is to instantiate clients with the password being $f(pwd)$, where the deterministic function f is specified by *Adv*. Only such an f that does not allow to check more than one password at a time may be allowed, and therefore strong restrictions on f are necessary. Indeed, setting $f(pwd) = 0$ on the first half of password domain D, and $f(pwd) = pwd$ on the second half, allows Adv to check half of password domain in one try. Restricting f to be a permutation does not work either, since applying such f allows to check whether pwd is a fixed point of f. Therefore functions f that have more than one or fewer than $|D| - 1$ fixed points are not allowed. At the same time, it is not hard to see that functions with $0, 1, |D| - 1$ or $|D|$ fixed points do not allow Adv to check more than one password at a time when server is running a secure protocol, and thus may be allowed in our definition. Indeed, a function with 0 fixed points always causes S_j^C to P$\perp$; one with 1 fixed point fp always causes P$\perp$, unless $p = pwd$, and thus allows to check precisely one password; one with $|D|$ fixed points (identity) never causes P$\perp$; one with $|D| - 1$ fixed points always succeeds, unless pwd is the non-fixed point, and thus allows to check precisely one password.

At the same time, the most natural mistyping functions (e.g. confusing the order of digits) do not satisfy the requirements on f and do help the adversary (e.g. Adv can quickly test if the pin consists of the same decimal digits). More generally, Adv may infer a lot from simply observing a large volume of traffic, noting the patterns of honest clients mistyping their passwords, and matching them with expected patterns. However, it is not clear how to analyze this advantage, so we choose not to include password mistypes in our model at all, with the understanding that protocol designers take this discussion into account.

This subtlety also arises in KE in the pure password model, when passwords need not be chosen uniformly from D. Indeed, let $D_1 \subset D$ be all elements of D that end with a 0, and $pwd \in D$ is chosen uniformly from D_1. Then a protocol Π that reveals pwd iff pwd is mistyped only in the last digit, would be secure under a natural definition that does not allow mistyping. This is because pwd would not be revealed, unless Adv already had tried it. At the same time, such protocol Π should not be deemed secure. We note that the recent definition of password based KE in the Universal Composability model ([24])

addresses the issue of mistyping by allowing the environment to both *choose* and *type* passwords.

On reporting failures to *Adv* immediately after failing. We justify our definitional decision to require that players don't have private failure outputs (either $\perp$ or P $\perp$), and *Adv* is informed of failure as soon as it output.

At first glance, it may seem that the ability of players to have private failure outputs may facilitate design of more efficient secure protocols. Indeed, consider a modification of (a vulnerable instance of) Π_{HK}, where, upon a password failure, the server does not report it to *Adv*, but produces a random key and simulates successful completion of KE. This change would have prevented our attack of Sect. 6.2. However, the achieved security would be illusory, since, in practice, it is hard to simulate successful completion well. Indeed, a P$\perp$ must be somehow registered and used by S. This changes the state of S (in particular, the counter of active attacks is incremented). Since C can login after $q-1$, but not after q P$\perp$'s, *Adv* is able to infer some information about S' outputs. Because of this "side channel", we choose not to allow private failure outputs in our definition. We further observe that the modification of Π_{HK} of this paragraph is insecure according to our definition. This is because the server does not report any P$\perp$'s.

We also note that protocol termination and failure reporting should be timely. Indeed, suppose S_j at some point "knows" he is going to output P$\perp$, that is, S_j entered a state from which all execution paths lead to outputting P$\perp$, and *Adv* learned this fact. Suppose S_j does not terminate yet, but is waiting to receive another message. Then *Adv* can delay the delivery of the message indefinitely, S_j would never report P$\perp$, and we don't count it. In particular, adding an extra round of communication to a secure protocol Π, in which parties say whether they failed, makes Π insecure. This is consistent with our desire to force a server to correctly and timely report active attacks.

6.4 Our Protocol

Let n be a security parameter. Let F be a PRFG and MAC be a secure message authentication code, as defined in Sect. 2.7.4 and 2.7.5. To simplify discussion, we present our constructions with the domains and ranges of PRFG and MAC equal to $\{0,1\}^n$. Let $E = (Gen, Enc, Dec)$ be a CCA secure public key encryption scheme, $F : \{0,1\}^n \times \{0,1\}^n \mapsto \{0,1\}^n$ be a PRFG, and $MAC : \{0,1\}^n \times \{0,1\}^* \mapsto \{0,1\}^n$ be a message authentication code. Let N_C be the name of the client C, drawn from $\{0,1\}^n$. Shorter names can be used for efficiency, if desired.

Consider the following KE protocol Π, with two types of players, a server S and a client C who have secretly agreed on a password $pwd \in_R D$, a long secret key $\ell \in_R \{0,1\}^n$. Also, S has generated public/private key pair (pk_S, sk_S), and gave pk_S to C.

Construction 14. *(KE in the Combined Key Model (Π).)*

S^C		C^S
choose $r \in_R \{0,1\}^n$		*choose* $k \in_R \{0,1\}^n$,
		set $\alpha = Enc_{pk_S}(N_C, pwd, k)$
	$r \rightarrow \cdots \leftarrow \alpha, MAC_\ell(\alpha)$	
verify $MAC_\ell(\alpha)$ *and* N_C;		*output*
if fail, output $\perp$ *and halt*		$K = F_k(r), sid = (r, \alpha)$
verify pwd;		
if fail, output $P\perp$ *and halt*		
else output		
$K = F_k(r), sid = (r, \alpha)$		

WLOG, we assume that all protocol messages are formed properly (i.e. values are drawn from the appropriate domains, etc.). Then a client instance never fails, while a server instance may. Note that Adv may cause non-partnered parties to output unrelated keys. This is not a problem (see Sect. 6.3.1 and Footnote 3).

We stress that the two flows of the protocol are independent, and thus either of the parties can be the initiator. The DoA attacks are prevented if *Adv* does not have ℓ, even though, in particular, *Adv* is able to resend old messages of the client. The latter causes a server to output a random (from the point of view of *Adv*) session key, thus *Adv* is not able to take advantage of it. This also does not enable *Adv* to "reset" the fail counter in real executions (and thus try many passwords undetected), since the same effect can be achieved by *Adv* executing a KE between honest S_j^C and C_i^S, and then cutting the communication.

We treat the policies of account suspension and resetting of failure counters as external to our discussion, but stress that care should be taken in designing and implementing them. In particular, the client's explicit consent (communicated over a secure session) should be necessary for resetting the failed attempts counter, since otherwise *Adv* can be undetected when trying passwords between legitimate client logins. A natural scenario would be that the server asks the client whether he mistyped the password a certain number of times, and when client confirms, the fail counter is reset.

We further note that we can prevent *Adv* from resending C's old replies to S (e.g. if it is undesirable to have "hanging" sessions) by including r in the encryption of the client's reply and adding the corresponding verification step to S. We chose not to include it because it disallows the independence of flows of KE, and it is unclear whether hanging sessions are "worse" than hanging KE.

An alert reader will notice that smart cards may be gainfully used in place of client's storage cards. A smart card may hide the long key ℓ, only exposing the MAC'ing interface. An interesting setting is when *Adv* can "borrow" and return (but not copy) the card, obtaining only a period of ability to MAC strings of his choice. Our protocol will not benefit from such security improvements: C's messages are independent of S's, and thus *Adv* can MAC all the strings he might possibly need for an attack (e.g. strings containing all possible passwords) in one batch. Again, including r in the encryption of C's reply resolves this problem.

Π **is secure.** We first observe that for every Adv_{sid} and Adv_{doa} playing games SID and DOA, their probability of winning is negligible. Indeed, in our protocol, partners never output different keys (since the session key is determined by sid). As for Adv_{doa}, for a server to output P$\perp$, it is necessary to forge a MAC on an encryption not produced by any of the honest clients. This is only possible with negligible probability without the knowledge of the long key ℓ, assuming security of MAC.

We formally consider the remaining games KE$_i$ and adversaries in Sect. 6.5. The structure of our proof is as follows. We start by reducing the KE adversaries to ones playing much simpler games. As a second step, we show that existence of new adversaries implies insecurity of either of the employed primitives. We consider several adversarial behaviours separately. Appendix 6.5.1 discusses the most interesting setting, where the adversary sees the long key and challenges a server instance, and we formally and carefully show the precise quantitative security of Π. Discussion of this section contains the main ideas of our entire proof. Sect. 6.5.2 addresses the remaining two games. Altogether, we've proven the following.

Theorem 15. *The protocol* Π *of Constr. 14 is a secure key exchange protocol in the combined keys model.*

On generalizing Constr. 14. Consider creating a family of protocols parameterized by a function f similarly to the approach of Halevi and Krawczyk. The goal is to shorten the plaintext of the encryption α sent by C, which may improve the performance of the protocol. We note that we already reduce the amount of data under the CCA-secure encryption – it is smaller than in any member of the HK families of KE protocols (but note that HK KE additionally achieve mutual authentication). We do not see how to further significantly increase efficiency by applying the HK idea to our protocols.

KE protocols for the HK setting. It is easy to see that removing the uses of the long key ℓ from the protocol of Constr. 14 casts it into the HK setting. The obtained protocol (explicated in Constr. 15 below) is a secure KE protocol in the HK setting, according to Def. 19. This conclusion immediately follows from the method of construction and Theorem 15.

Construction 15. *(KE in the HK setting.)*

S^C		C^S
choose $r \in_R \{0,1\}^n$		*choose* $k \in_R \{0,1\}^n$,
		set $\alpha = Enc_{pk_S}(N_C, pwd, k)$
	$r \rightarrow \cdots \leftarrow \alpha$	
verify N_C*;*		*output*
if fail, output $\perp$ *and halt*		$K = F_k(r), sid = (r, \alpha)$
verify pwd;		
if fail, output $P\perp$ *and halt*		
else output		
$K = F_k(r), sid = (r, \alpha)$		

Protection against the compromise of the server's password file. Recall that we previously assumed that the server's private information is never compromised. We now briefly discuss how storing passwords in a hashed form helps maintain a reasonable level of security even if *Adv* steals the password file. This discussion is informal, since, even though a password may be forced to be sufficiently long (40-60 bits), due to human memory limitations, it often contains only a small amount of entropy. Consequently, it is hard, if at all possible, to formally justify the advantage of the server storing hashes of passwords instead of their plaintext values. Indeed, if *Adv* steals a file of hashed user passwords (or any other information allowing S to verify a password), he can compute the corresponding passwords in polytime.

From the another point of view, if *Adv* does not know a client C, C's password may be long and unpredictable to *Adv*. Indeed, it might include many easy to remember personal references, which *Adv* does not know. Thus, the passwords may be viewed as having some probability of being chosen from a low-entropy distribution, if the user fits the profile that *Adv* has. Otherwise, the passwords may be chosen from a high-entropy distribution. In this view, resilience to server compromise is meant to protect the "strong" passwords.

Moreover, in practice it is often unclear how to exploit the even the relatively low entropy of passwords. See Narayanan and Shmatikov [77] for recent results and background in password cracking. Further, most of the attacks (including that of [77]) employ expensive precomputation, after which they can attack multiple passwords at a much lower cost per password. Recall, the benefits of precomputation are removed by "salting", which can be viewed as using a different hash function for each user's password. In addition, slow hash functions may be used to increase the cost of the attack, as discussed in [20] and is done in the UNIX `crypt()` implementation.

Thus, storing passwords only in the salted hashed form on the server seems to provide additional significant level of protection, at least with the current state of the art of password cracking. We note that our protocols can be trivially modified to allow for this second line of defense, e.g., as follows. The server S will store a randomly chosen salt s and a hash $h = H(pwd, s)$, instead of the C's password pwd. The client C stores s on the card. When computing α above, C includes $H(pwd, s)$ instead of the plaintext pwd. The password verification procedure of S^C is amended correspondingly. We envision the above modifications for most practical situations. As previously discussed, other second-line defense techniques, e.g. those described in [58], can be used to also achieve heuristic security against the compromise of the secret key of the server. Finally, we note that a compromise of the long key ℓ of the client (which is also stored on the server) is already addressed in our definition by allowing *Adv* to steal ℓ.

How to change passwords. In practice, throughout the life cycle of a client-server system, it might be necessary to change passwords of clients. Usually, in the KE literature this need is treated as external to the definition and protocols. It turns out that in our setting it requires special care. We now briefly describe the subtle problem and informally suggest several solutions.

Suppose client C securely (e.g. in a private meeting with the server S) changes his password from pwd to pwd'. Then Adv can perform a DoA attack by simply sending to S^C C's old messages, containing (properly encrypted and MAC'ed) pwd. Intuitively, the problem arises from the fact that only a part of the key of C is modified when the password is updated. In other words, clients with related credentials would exist in the system, violating our assumption on the independence of key generation of players (see first item in Sect. 6.3.3 for more discussion).

A natural solution is to disallow password-only updates to credentials, and to require regeneration of the long key ℓ as well. Such updates will not cause problems, since the new key (pwd', ℓ') is fully independent from the old one, and all previous transcripts obtained by Adv become useless[4].

We also note that it is possible to allow password-only updates, at the cost of complicating the protocol (and the definition). This may be desirable when updates of the client's storage are inconvenient or costly. Security in this setting can be achieved, for example, by a modification of our protocol into a challenge-response one. Alternatively, it is possible to preserve the property of independence of flows in the KE protocol. This can be done at the cost of keeping (and appropriately using) password update counters by both S and C[5].

Our definitions also would need to be modified if password-only updates are allowed. Indeed, our protocol of Constr. 14 is secure according to Def. 18, yet it is clearly vulnerable to the DoA attack in the current setting. We leave the resulting update to the definitions outside the scope of this work.

[4]Note a technicality that incomplete instances S^C and C^S should be terminated at the time of key update.

[5]While storage on the server side is readily available, we may have a read-only client's storage medium. If so, the counter may be considered as part of the password, with the assumption that the upper bound on the number of password updates is small. We stress that in this case, a non-matching counter value will cause S^C output $\perp$, and not P$\perp$.

6.5 Proof of security of the protocol of Constr. 14 (Theorem 15)

We first prove that, assuming security of the underlying primitives of Π, there does not exist an adversary winning the game KE_1 too often. The proof of this case is delicate due to handling precise quantitative advantage of *Adv*; it presents main ideas for the proof of the other cases.

Proposition 1. *If the PRFG F and the CCA encryption scheme E used in Π are secure, then for every polytime Adv, the probability p of Adv winning the game KE_1 is no more than $1/2 + \frac{q}{2|D|} + \epsilon$, where ϵ is negligibly small (in the security parameter n).*

Prop. 1 follows from lemmas 2 and 3, presented in Sect. 6.5.1.

The other cases are handled by

Proposition 2. *If the PRFG F, MAC, and the CCA encryption scheme E used in Π are secure, then for every polytime Adv_1 and Adv_2 the probabilities of them winning the games KE_2 and KE_3 respectively are no more than $p > 1/2+\epsilon$, where ϵ is negligibly small (in the security parameter n).*

Prop. 2 follows from lemmas 4, 5 and 6, presented in Sect. 6.5.2.

Theorem 15 follows from Prop. 1 and 2.

6.5.1 Proof for the case when the adversary is given the long key and challenges the server

Consider the following game (parameterized by n). that a distinguisher $Dist_1$ plays. We suggest looking at the game briefly at the first reading – the motivation behind it would be clear in the proof of the reduction from game KE_1 (Lemma 2).

Game G_1. *A maximum number of "password tries" q is deterministically (based on n) chosen by $Dist_1$ and fixed. The game initializes a CCA secure encryption scheme* *(by generating public and private keys pk_S and sk_S) and randomly chooses the password $pwd \in_R D$. Only the public key pk_S is given to $Dist_1$. $Dist_1$ queries the* decryption oracle $O_D(e') = Dec_{sk_S}(e')$ *to obtain decryptions of chosen strings. Then $Dist_1$ chooses a "client name" N_C. Then, for $i = 1, ..., u$, $Dist_1$ queries the* encryption oracle O_E *that produces random encryptions $e_i = Enc_{pk_S}(N_C, pwd, k_i)$, where $k_i \in_R \{0,1\}^n$ are chosen randomly and unknown to $Dist_1$. Here u is chosen by $Dist_1$. Then $Dist_1$ proceeds by executing Steps 1 - 2 multiple times, in any order:*

1. *$Dist_1$ queries the* PRFG oracle $O_F(i, r) = F_{k_i}(r)$, *where k_i was chosen (but not revealed) by O_E during it's i-th query. Here $r \in \{0,1\}^n$ and $i \in \{1..u\}$ are chosen by $Dist_1$.*

2. *$Dist_1$ queries the* decryption oracle $O_D(e')$, *where e' is chosen by $Dist_1$. He is not allowed to query O_D on any e_i obtained from O_E.*

Then $Dist_1$ chooses $i \in \{1, ..., u\}$ and $r_0 \in \{0,1\}^n$ and queries the challenge oracle $O_C(i, r_0)$. *O_C produces a challenge as follows: it randomly chooses a bit b and a string $\rho \in_R \{0,1\}^n$. Then $O_C(i, r_0) = F_{k_i}(r_0)$ if $b = 0$, and $O_C(i, r_0) = \rho$ if $b = 1$. $Dist_1$ is not allowed to query $O_C(i, r_0)$, if he queried $O_F(i, r_0)$.*

Then, $Dist_1$ continues running Steps 1-2, with the exception that he is not allowed to query $O_F(i, r_0)$.

Finally, $Dist_1$ generates a list of q password guesses $PL = \{p_1, ..., p_q\}$ and outputs a bit b'. $Dist_1$ wins if $pwd \in PL$ or if $b = b'$.

Lemma 2. *Suppose there exists an adversary Adv that asks for the long key ℓ, always challenges a server instance, and breaks the protocol Π. Then there exists $Dist_1$ winning the game G_1 with probability non-negligibly greater than $1/2 + \frac{q}{2|D|}$, where G_1 is run with the same encryption scheme E and PRFG F as Π.*

Proof. We prove the theorem by constructing $Dist_1$ that wins G_1, essentially whenever Adv wins the KE game. $Dist_1$ simulates an environment (i.e. KE players and their actions), in which he runs *Adv*, answers *Adv*'s queries and uses *Adv*'s decisions to make decisions in G_1. We say "$Dist_1$ stops", meaning "$Dist_1$ finishes processing *Adv*'s request and returns control to *Adv*", and "$Dist_1$ sends (outputs) m", meaning "$Dist_1$ simulates the given player sending (outputting) m, by giving m to *Adv*".

$Dist_1$ starts up *Adv*, who outputs the threshold q and requests to create (the only) server S. $Dist_1$ then starts the game G_1 with q, and obtains the public key pk_S for *Enc*. $Dist_1$ sends pk_S to *Adv* as the public key of the server. $Dist_1$ initializes its password list PL to empty.

$Dist_1$ then runs *Adv* and satisfies its requests for information as follows. Note that a client C must have been created to create its instances C_i or server instances S_j^C.

1. *Adv creates a bad client* B^i*:*

   Adv chooses the password and the long key, and reveals them to S (thus giving them to $Dist_1$).

2. *Adv creates (the only) honest client* C *with the name* N_C*:*

   $Dist_1$ chooses the name N_C for G_1 to be the name of the client. Let u be the upper bound on the number of client instances *Adv* creates. Then, for $i = 1, ..., u$, $Dist_1$ queries oracle O_E and obtains random encryptions $e_i = Enc_{pk_S}(N_C, pwd, k_i)$, where $k_i \in_R \{0,1\}^n$ are chosen randomly and are unknown to $Dist_1$. (We note that *Adv* did not cause any calls to O_F or O_C yet, although he may have created and run server with corrupt clients. Therefore, there is no conflict with G_1's scheduling.) Then $Dist_1$ randomly chooses $\ell \in_R \{0,1\}^n$ to be C's long key. *Adv* asks for it, so $Dist_1$ reveals ℓ to *Adv*.

3. *Adv creates an instance* S_j^C *or* $S_j^{B^i}$ *of* S *and starts the protocol:*

   $Dist_1$ randomly chooses $r_j \in_R \{0,1\}^n$ and sends it.

4. *Adv creates new (i-th) instance* C_i *of the honest client* C*.*

   Recall that $Dist_1$ already obtained e_i from O_E. $Dist_1$ computes $mac_i = MAC_\ell(e_i)$

and sends (e_i, mac_i).

5. *Adv delivers a message m_{C_i} to an instance C_i of honest client C (allegedly) from server S:*

   $Dist_1$ gives to Adv the session id $sid_i = (m_{C_i}, e_i)$. Recall, e_i is the encryption previously sent by C_i.

6. *Adv delivers a message $m_{S_j} = (e', m')$ to S_j^C (allegedly) from client C (recall, C is honest):*

   If $m' \neq MAC_\ell(e')$, $Dist_1$ outputs $\perp$and stops. Otherwise $Dist_1$ proceeds as follows.

   If $e' = e_i$ was obtained from O_E, $Dist_1$ gives to Adv the session id $sid_j = (r_j, e_i)$. Recall, r_j is the message previously sent by S_j^C.

   Otherwise, if e' was not obtained from O_E, $Dist_1$ continues and decrypts e' by querying the decryption oracle $O_D(e')$ to obtain (N'_C, pwd', k'). If $N'_C \neq N_C$, $Dist_1$ outputs $\perp$and stops. Otherwise, i.e. if the client's name matches, $Dist_1$ adds pwd' to the list PL of passwords to try, unless this causes $|PL| > q$. (Since Adv cannot communicate with S_j^C after q P$\perp$'s, the only case when Adv causes the $q+1$-st execution of this clause is when Adv had produced a valid guess at C's password. If so, pwd has already been added to PL, and there is no benefit in adding anything to PL.) Finally, $Dist_1$ outputs P$\perp$ and stops. (Note if this response is incorrect, then pwd has been added to PL, and $Dist_1$ wins, so we don't worry about properly simulating the game anymore.)

7. *Adv delivers a message $m_{S_j} = (e', m')$ to $S_j^{B^i}$ (allegedly) from client B^i(recall, B^i is corrupt):*

   Recall that $Dist_1$ knows B^i's long key and password. $Dist_1$ verifies MAC; if verification fails, $Dist_1$ outputs $\perp$and stops. If $e' = e_i$ was obtained by any oracle call to O_E, $Dist_1$outputs $\perp$and stops(since the client name would not verify[6].)

[6]This conclusion cannot be made when attempting to reduce the KE game of the Halevi-Krawczyk

Otherwise (if MAC checked and e' was not obtained from O_E) $Dist_1$ proceeds as follows. $Dist_1$ decrypts e' by querying the decryption oracle $O_D(e') = (N'_C, pwd', k')$ and acts according to the Server's protocol, as follows. $Dist_1$ verifies whether N'_C equals to the name of B^i. If not, $Dist_1$ outputs $\perp$and stops. Then $Dist_1$ verifies whether pwd' is the B^i's password; if not, $Dist_1$ outputs P$\perp$ and stops. Otherwise, $Dist_1$ gives to Adv the session id $sid_j = (r_j, e')$.

8. *Adv sends an* open *request on a (completed and not failed or challenged) client instance C_i of C:*

   Note that C_i output $sid_i = (m_{C_i}, e_i)$. $Dist_1$ queries oracle $O_F(i, m_{C_i}) = F_{k_i}(m_{C_i})$, and gives the answer to Adv. Note that there are restrictions on when $Dist_1$ is allowed to call O_F (O_F and O_C cannot be called with the same parameters). We argue later that we are not violating them.

9. *Adv sends an* open *request on a (completed and not failed or challenged) server instance S_j of S:*

   Recall that S_j received $m_{S_j} = (e', m')$ and S_j output $sid_j = (r_j, e')$. If $e' = e_i$ was generated by O_E, then $Dist_1$ queries oracle $O_F(i, r_j)$ and outputs the answer. As in 8, we will later argue that we are not violating G_1's restrictions.

   Otherwise, $Dist_1$ decrypts e' by calling $O_D(e')$ and outputs $F_{k'}(r_j)$, where k' is the key inside e'. Note that this is the case corresponding to the last paragraph of case 7 above, since $Dist_1$ always reports failure when S_j^C receives e' not generated by O_E. No O_F call is made in this clause.

10. *Adv sends a* challenge *request on a (completed and not failed or opened) server instance S_j^C of S:*

    Recall, S_j^C sent r_j, received $m_{S_j} = (e', m')$ and output $sid_j = (r_j, e')$. If $e' = e_i$

protocol to G_1 in a natural way, and thus $Dist_1$ cannot answer correctly without calling $O_D(e_i)$. However, it is crucial that $O_D(e_i)$ is not called here.

was generated by O_E (i.e. sent by a client C_i), $Dist_1$ queries the challenge oracle $ch = O_C(i, r_j)$, gives ch to Adv and (later, after submitting the list PL) submits Adv's output as his answer to the challenge of G_1. As in 8 and 9, we will later argue that we are not violating G_1's restrictions when querying $O_C(i, r_j)$.

Note that the case when e' of m_{S_j} was not generated by O_E cannot happen, since $Dist_1$ would have reported to Adv that S_j^C failed.

We note that $Dist_1$ always ensures legality of calls to $O_D(e)$ by checking that e was not generated by O_E. We now argue that all calls to O_F in 8–9, and to O_C in 10 will be legal requests in G_1, that is that $Dist_1$ never calls both $O_F(i, r)$ and $O_C(i, r)$, for any pair (i, r).

First note that O_F and O_C are only called when Adv opens or challenges instances, respectively. Adv always challenges a server instance. Suppose, he challenged S_j^C, and thus caused the call $O_C(i, r_j)$, where e_i was generated by O_E and sent by some client C_i. Consider two possible cases. First, for $k \neq j$, Adv opens (either earlier or later) a server instance S_k, causing a call $O_F(i', r_k)$. This call is legal, since $Prob(r_j = r_k)$ is negligible. Second, Adv opens a client instance C_k^S, thus causing a call $O_F(k, m_{C_k})$. Suppose this call is illegal, i.e. $i = k$ (implying that $e_i = e_k$) and $r_j = m_{C_k}$. However, in this case, the session ids output by the parties match. Then S_j^C and C_k^S are partners, and such Adv's behaviour is not allowed in KE$_1$.

Now it is easy to see that the simulated messages provided by $Dist_1$ are distributed almost identically to those generated in a real execution, until the point when Adv does guess the password correctly, and $Dist_1$ incorrectly returns P$\perp$. What happens after that point, however, does not matter, since $Dist_1$ had already won the game.

By assumption of the lemma, Adv wins with probability non-negligibly more than $1/2 + \frac{q}{2|D|}$. It is easy to see that $Dist_1$ wins whenever Adv wins, except for a negligible fraction of the time. Therefore, the constructed $Dist_1$ wins the game G_1 with probability non-negligibly more than $1/2 + \frac{q}{2|D|}$.

□

We now show that the adversary $Dist_1$ described in Lemma 2 cannot exist, if secure primitives are used.

Lemma 3. *If the PRFG F and the CCA encryption scheme E used in G_1 are secure, then for every polytime $Dist_1$, the probability p of $Dist_1$ winning the game G_1 is no more than $1/2 + \frac{q}{2|D|} + \epsilon$, where ϵ is negligibly small (in the security parameter n).*

Proof. Consider a polytime $Dist_1$. We first argue that he cannot produce a password list PL containing pwd with probability significantly more than $q/|D|$. To prove this, we strengthen $Dist_1$ by allowing him choose k_i used in the calls to O_E. Then G_1 can be further simplified – $Dist_1$ does not need access to O_F (he can evaluate it himself). It is now easy to see that if $Dist_1$ can produce a list PL of q passwords with probability significantly more than $q/|D|$, he can be used to break the security of E (since he must have obtained some information about pwd from playing essentially the game of the CCA security.)

Now, return to the original $Dist_1$ and G_1. Let E_1 be the event of $Dist_1$ producing PL containing pwd, and E_2 be the event of $Dist_1$ winning by answering the challenge correctly. Then the probability of $Dist_1$ winning G_1 is $p = prob(E_1) + (1 - prob(E_1))prob(E_2|\neg E_1)$. Note that the lemma trivially holds for n, where $q \geq |D|$.

From now on, consider n, such that $q < |D|$ (q is polynomially bounded). Then, $Prob(\neg E_1)$ is bounded away from 0 by a polynomial (in n) fraction. We now show that for $Dist_1$, $prob(E_2|\neg E_1) < 1/2 + \epsilon_2$, where ϵ_2 is negligible. Suppose otherwise. Then we construct a polytime D' who, with the knowledge of pwd, answers the challenge of G_1 with probability significantly better than 1/2. D' proceeds as follows. He runs $Dist_1$ up to the point when $Dist_1$ produces PL. D' checks whether $pwd \in PL$. If so, he flips a coin to answer the challenge. If not (and this happens non-negligibly often), he continues running $Dist_1$ (and obtains non-negligible advantage). At the same time, it can be easily shown by standard hybrid techniques that such D' cannot exist. Thus $prob(E_2|\neg E_1) < 1/2 + \epsilon_1$.

Therefore, if all the employed primitives are secure,

$$p = prob(E_1) + (1 - prob(E_1))prob(E_2|\neg E_1) < \frac{q}{|D|} + \epsilon_1 + (1 - \frac{q}{|D|})(1/2 + \epsilon_2) = 1/2 + \frac{q}{|D|} + \epsilon.$$

□

6.5.2 Other cases

In all other cases, we reduce the KE game to a simpler variant G_2 of the game G_1.

Game G_2**.** *G_2 proceeds exactly as G_1 with the following two exceptions. First, the client's password pwd is revealed to the distinguisher $Dist_2$ as soon as $Dist_2$ sets the name C. Second, $Dist_2$ is not allowed to win by presenting PL (thus PL generation is omitted).*

Lemma 4. *If there exists an adversary Adv breaking the protocol Π that challenges a client and is given the long key ℓ and the password pwd, then there exists $Dist_2$ winning the game G_2 with probability non-negligibly greater than $1/2$.*

Proof. The construction of $Dist_2$ and the following discussion proceed almost identically to construction of $Dist_1$ of Lemma 2. Here we only point out the differences in construction and discussion.

- PL is not created nor used in any way by $Dist_2$.
- In Step 2, when the honest client is created, both the long key ℓ and the password pwd (obtained from G_2) are given to Adv.
- In Step 6 (Adv delivers a message $m_{S_j} = (e', m')$ to S_j^C (allegedly) from client C) $Dist_2$ proceeds like $Dist_1$, with the following exception. If e' (an encryption of (N'_C, pwd', k')), was not obtained from O_E, and the client name matches ($N'_C = N_C$), then instead of modifying PL, $Dist_2$ does the following. Recall, $Dist_2$ knows

the password pwd of C. If $pwd' \neq pwd$, $Dist_2$ outputs P$\perp$, otherwise $Dist_2$ outputs $sid = (r_j, e')$. Recall, r_j is the message previously sent by S_j^C.

- $Dist_2$ handles a new type of request: *Adv sends a* challenge *request on a (completed and not failed or opened) client instance* C_i^S *of* C*:*

  Note that C_i^S previously received m_{C_i} and output $sid_i = (m_{C_i}, e_i)$. $Dist_2$ queries the challenge oracle $ch = O_C(i, m_{C_i})$, gives ch to Adv and submits Adv's output as the answer to the challenge of G_2. Note that there are restrictions on when $Dist_2$ is allowed to call O_C (O_F and O_C cannot be called with the same parameters). We argue later that we are not violating them.

- Request 10 (challenging a server instance) is now not allowed.

We note that all oracle calls made by $Dist_2$ are legal requests in G_2. The argument is also similar to that of Lemma 2. Indeed, as in construction of $Dist_1$, we always ensure that e was not generated by O_E before calling $O_D(e)$.

Further, O_F and O_C are only called when Adv opens or challenges instances, respectively. Consider the two possible cases (there are only two since Adv always challenges a client). First, Adv opened and challenged client instances C_{i_1} and C_{i_2}. Then, for the conflict to happen, it must be that $e_{i_1} = e_{i_2}$, which happens with negligible probability. Second, Adv opened a server instance S_j^C and a challenged a client instance C_i^S. For the conflict to happen, it must be that the client instance received r_j, and the server instance received e_i during the game. However, in this case, the sid output by the instances would match, and thus C_i and S_j would be partners, and Adv would not be allowed to challenge C_i^S and open S_j.

Now it is easy to see that the simulated messages provided by $Dist_2$ are distributed almost identically to those generated in a real execution. By assumption of the lemma, Adv wins with probability non-negligibly more than 1/2. It is easy to see that case $Dist_2$ wins whenever Adv wins, except for the negligible fraction of the time. Therefore, the constructed $Dist_2$ wins the game G_2 with probability non-negligibly more than 1/2.

□

Finally, we consider the adversary who is not given the long key ℓ, and is attacking the server.

Lemma 5. *Suppose the employed MAC scheme is secure. Then, if there exists an adversary Adv breaking the protocol* Π *who is not given the long key* ℓ *and is attacking the server, then there exists* $Dist_2$ *winning the game* G_2 *with probability non-negligibly greater than* 1/2.

Proof. The construction of $Dist_2$ and the following discussion proceed almost identically to construction of $Dist_1$ of Lemma 2. Here we only point out the differences in construction and discussion.

- PL is not created nor used in any way by $Dist_2$.
- In Step 2, when the honest client is created, the long key ℓ is not revealed to Adv. The password pwd (obtained from G_2) is given to Adv.
- In Step 6 (Adv delivers a message $m_{S_j} = (e', m')$ to S_j^C (allegedly) from client C) $Dist_2$ proceeds like $Dist_1$. We note that e' was not obtained from O_E only with negligible probability (since otherwise we can construct a forger for MAC), and thus we don't handle the corresponding clause.
- In Step 10 (Adv sends a *challenge* request on a (completed and not failed or opened) server instance S_j^C of S:) $Dist_2$ proceeds like $Dist_1$. (Note that e' was not obtained from O_E only with negligible probability, due to the security of MAC; thus we don't handle the corresponding clause.)

We note that all oracle calls made by $Dist_2$ are legal requests in G_2. The argument is analogous to that of Lemma 2. Thus, the simulated messages provided by $Dist_2$ are distributed almost identically to those *Adv* sees in a real execution. By assumption of the lemma, *Adv* wins with probability non-negligibly more than 1/2. It is easy to see that case $Dist_2$ wins whenever *Adv* wins, except for a negligible fraction of the time. Therefore, the constructed $Dist_2$ wins the game G_2 with probability non-negligibly more than 1/2.

□

We now show that the adversary described in Lemmas 4 and 5 cannot exist, if secure schemes are used.

Lemma 6. *If the PRFG F and the CCA encryption scheme E used in G_2 are secure, then there does not exist a polytime $Dist_2$ winning the game G_2 with probability $p > 1/2 + \delta$, where δ is not negligibly small (in the security parameter n).*

The proof of Lemma 6 is done by a standard hybrid argument, and is omitted.

□

Chapter 7

Summary and Future Work

In this dissertation, we presented our results in the areas of SFE and KE. We presented new, more efficient protocols for secure evaluation of variants and extensions of the GT functionality, such as auctions. We presented a new, more efficient information-theoretic reduction of secure evaluation of any boolean formula to OT. Finally, we discussed methods of securing network communication between parties authenticated by a combination of different types of credentials.

Presented work and background discussion highlight some of the research directions of modern cryptography and give a flavour of the problems that arise in these areas. While we have shown some aspects of securing computation and communication of parties, we did not consider many important issues and settings. We now give a brief overview of research directions that naturally follow from or related to what we discussed in this dissertation.

Secure Multi-Party Computation

Most of our work addresses two-party computation. We went slightly beyond the two-party setting in Chapter 4, where we considered a helping semi-honest server. The general setting with an arbitrary number of parties is a natural interesting generalization of our setting. Current state of the art of information-theoretic multi-party evaluation of formulas has complexity $\Theta(2^d 2^{\Theta(\sqrt{d})})$ (see, e.g. Cramer, Fehr, Ishai and Kushilevitz [30]). It seems that the techniques developed in Chapter 5 can be applied to the multi-party setting as well. If the overhead of this application is less than exponential (which seems likely), then this would imply new upper bounds on general SFE.

Non-interactive blind computing

Consider the setting where the Server S wishes to evaluate the function of the Client C on S's data. C only learns the value of the function, and S learns nothing (not even the function itself). This can be done by a secure evaluation of the universal circuit [88], for example, as described in [86], and this is currently the best known method. Using universal circuits, however, has a high overhead. One of the properties of our GESS construction of Chapter 5 is that the evaluation of the gates of the formula is independent of the gate semantics. It seems possible to exploit this property together with "mangling" the wire connections of the circuit to construct more efficient solutions to this problem.

Security of Composition of Protocols

Until very recently, in proofs of security, the execution of protocols was assumed to be performed in relative isolation. However, in real life, a single person may simultaneously chat, shop, bank, trade, play, and vote online. Each of these activities may require credentials. These credentials may be correlated for the same person, further violating the assumption that protocols are run in isolation. Moreover, failures or poor design of some applications may affect security of other applications run by the same person. Today's attempts to consider security of execution in an arbitrary environment, e.g. [23, 79, 3], are still in an early stage, and significant theoretical and practical research effort is needed to resolve them.

KE

We considered KE in the specific “combined keys” setting. A number of natural generalizations of this model is possible. For example, it is interesting to consider the setting, where, instead of the storage card, the client is given a stealable but tamper-proof smart card. We can now make an assumption that the smart card and the keys it holds cannot be copied, and the stored keys are accessed indirectly via, for example, a signing interface. Such an assumption would allow for the design of more efficient and more secure protocol that make use of this new security feature of the storage medium.

It also seems to be interesting to address the issue of password mistyping in the definition. Recall, in our discussion in Sect. 6.3.3, we showed that in our framework it is not possible to model security in the presence of password mistyping. At the same time, the more complicated Universally Composable-style of definition [23] allows for this generality. It would be interesting to devise definitions that do not rely on the framework of universal composability and handle password mistyping.

Bibliography

[1] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *Proc. EUROCRYPT 2001*, pages 119–135, 2001.

[2] Joy Algesheimer, Christian Cachin, Jan Camenisch, and Gunter Karjoth. Cryptographic security for mobile code. In *SP '01: Proceedings of the IEEE Symposium on Security and Privacy*, page 2. IEEE Computer Society, 2001.

[3] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In M. Naor, editor, *First Theory of Cryptography Conference, TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354. Springer-Verlag, 2004.

[4] D A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In *Proc.* 18*th ACM Symp. on Theory of Computing*, pages 1–5, New York, NY, USA, 1986. ACM Press.

[5] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proc.* 22*nd ACM Symp. on Theory of Computing*, pages 503–513, 1990.

[6] Donald Beaver. Minimal-latency secure function evaluation. In *Proc. EUROCRYPT 2000*, pages 335–350. Springer, 2000. Lecture Notes in Computer Science, vol. 1807.

[7] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract).

In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 419–428, New York, NY, USA, 1998. ACM Press.

[8] Mihir Bellare, Oded Goldreich, and Anton Mityagin. The power of verification queries in message authentication and authenticated encryption. Cryptology ePrint Archive, Report 2004/309, 2004. `http://eprint.iacr.org/`.

[9] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 547–557, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[10] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000*, pages 139–155, 2000.

[11] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO '93: Proceedings of the 13th annual international cryptology conference on Advances in cryptology*, pages 232–249, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[12] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secureagainst dictionary attacks. In *SP '92: Proceedings of the 1992 IEEE Symposium on Security and Privacy*, page 72, Washington, DC, USA, 1992. IEEE Computer Society.

[13] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[14] Ian F. Blake and Vladimir Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *Advances in Cryptology - ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 515–529. Springer, 2004.

[15] Ian F. Blake and Vladimir Kolesnikov. Conditional encrypted mapping and comparing encrypted numbers. In *Financial Cryptography and Data Security Conference 2006*, volume ?? of *Lecture Notes in Computer Science*, pages ??–?? Springer, 2006.

[16] Maria Luisa Bonet and Samuel R. Buss. Size-depth tradeoff for boolean formulae. *Information Processing Letters*, 11:151–155, 1994.

[17] Maurizio Kliban Boyarsky. Public-key cryptography and password protocols: the multi-user case. In *CCS '99: Proceedings of the 6th ACM conference on Computer and communications security*, pages 63–72, New York, NY, USA, 1999. ACM Press.

[18] V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password-Authenticated Key Exchange Using Diffie-hellman. In B. Preneel, editor, *Proceedings EUROCRYPT 2000*, pages 156–171, 2000.

[19] N. H. Bshouty, R. Cleve, and W. Eberly. Size-depth tradeoffs for algebraic formulae. In *Proc. 32nd IEEE Symp. on Foundations of Comp. Science*, pages 334–341. IEEE, 1991.

[20] Samuel R. Buss and Peter N. Yianilos. Secure short key cryptosystems: 40 bits are enough. Technical report, NEC Research Institute, Princeton, NJ, 1999.

[21] Christian Cachin. Efficient private bidding and auctions with an oblivious third party. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 120–127. ACM Press, 1999.

[22] Christian Cachin, Jan Camenisch, Joe Kilian, and Joy Muller. One-round secure computation and secure autonomous mobile agents. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, 2000.

[23] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/`.

[24] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In *EUROCRYPT 2005*, pages 404–421, 2005.

[25] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 453–474, London, UK, 2001. Springer-Verlag.

[26] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In *EUROCRYPT '02: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, pages 337–351, London, UK, 2002. Springer-Verlag.

[27] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.

[28] T. Clancy. Eap password authenticated exchange, draft archive. http://www.cs.umd.edu/ clancy/eap-pax/, 2005.

[29] R. Cleve. Towards optimal simulations of formulas by bounded-width programs. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 271–277, New York, NY, USA, 1990. ACM Press.

[30] Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multi-party computation over rings. In *Proc. EUROCRYPT 2003*, pages 596–613, 2003.

[31] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, pages 13–25, 1998.

[32] G. Di Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and time-released encryption. In *Proc. CRYPTO 99*, pages 74–89. Springer-Verlag, 1999. Lecture Notes in Computer Science, vol. 1592.

[33] Giovanni Di Crescenzo. Private selective payment protocols. In *Financial Cryptography*, pages 72–89, 2000.

[34] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[35] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 542–552. ACM Press, 1991.

[36] J. H. Ellis. The history of non-secret encryption. `http://www.cesg.gov.uk/site/publications/media/ellis.pdf`.

[37] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[38] Uri Feige, Joe Killian, and Moni Naor. A minimal model for secure computation (extended abstract). In *Proc. 26th ACM Symp. on Theory of Computing*, pages 554–563. ACM Press, 1994.

[39] Joan Feigenbaum. Encrypting problem instances: Or ..., can you take advantage of someone without having to trust him? In *Proc. CRYPTO 85*, pages 477–488, 1985.

[40] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.

[41] Marc Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *RSA Security 2001 Cryptographer's Track*, pages 457–471. Springer-Verlag, 2001. Lecture Notes in Computer Science, vol. 2020.

[42] Internet Engineering Task Force. Eap password authenticated exchange. http://www.ietf.org/internet-drafts/draft-clancy-eap-pax-03.txt, 2005.

[43] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Proc. EUROCRYPT 2004*, pages 1–19. Springer-Verlag, 2004. Lecture Notes in Computer Science, vol. 3027.

[44] Steven D. Galbraith. Elliptic curve paillier schemes. *Journal of Cryptology*, 15(2):129–138, 2002.

[45] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[46] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 151–160, New York, NY, USA, 1998. ACM Press.

[47] Oliver Giel. Branching program size is almost linear in formula size. *J. Comput. Syst. Sci.*, 63(2):222–235, 2001.

[48] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM Press.

[49] Oded Goldreich. *Foundations of Cryptography*, volume 2: Basic Applications. Cambridge University Press, 2004.

[50] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.

[51] Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 408–432, London, UK, 2001. Springer-Verlag.

[52] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *FOCS*, pages 174–187, 1986.

[53] S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proc.* 14*th ACM Symp. on Theory of Computing*, pages 365–377, San Francisco, 1982. ACM.

[54] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[55] L. Gong, M. A. Lomas, R. M. Needham, and J. H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.

[56] Iftach Haitner. Implementing oblivious transfer using collection of dense trapdoor permutations. In *First Theory of Cryptography Conference, TCC 2004, Proceedings*, pages 394–409, 2004.

[57] Shai Halevi and Hugo Krawczyk. Public-key cryptography and password protocols. In *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security*, pages 122–131, New York, NY, USA, 1998. ACM Press.

[58] Shai Halevi and Hugo Krawczyk. Public-key cryptography and password protocols. *ACM Trans. Inf. Syst. Secur.*, 2(3):230–268, 1999.

[59] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *ISTCS '97: Proceedings of the Fifth Israel Symposium on the Theory of Computing Systems (ISTCS '97)*, page 174, Washington, DC, USA, 1997. IEEE Computer Society.

[60] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Proc. 41th IEEE Symp. on Foundations of Comp. Science*, page 294. IEEE Computer Society, 2000.

[61] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

[62] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02)*, 2002.

[63] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 20–31, Chicago, 1988. ACM.

[64] Vladimir Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In *Advances in Cryptology - ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 136–155. Springer, 2005.

[65] Vladimir Kolesnikov and Charles Rackoff. Key exchange using passwords and long keys. In *Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 5-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 100–119. Springer, 2006.

[66] H. Krawczyk. Skeme: a versatile secure key exchange mechanism for internet. In *SNDSS '96: Proceedings of the 1996 Symposium on Network and Distributed System Security (SNDSS '96)*, page 114, Washington, DC, USA, 1996. IEEE Computer Society.

[67] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. In *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 109–118, New York, NY, USA, 2006. ACM Press.

[68] Sven Laur and Helger Lipmaa. Additive conditional disclosure of secrets and applications. Cryptology ePrint Archive, Report 2005/378, 2005. `http://eprint.iacr.org/`.

[69] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Proc. CRYPTO 00*, pages 20–24. Springer-Verlag, 2000. Lecture Notes in Computer Science, vol. 1880.

[70] Yehuda Lindell and Benny Pinkas. A proof of Yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004. `http://eprint.iacr.org/`.

[71] Tsutomu Matsumoto, Koki Kato, and Hideki Imai. Speeding up secret computations with insecure auxiliary devices. In *Proc. CRYPTO 88*, pages 497–506, 1988.

[72] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437, New York, NY, USA, 1990. ACM Press.

[73] Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *STOC '01: Proceedings of the thirty-third annual ACM sym-*

posium on Theory of computing, pages 590–599, New York, NY, USA, 2001. ACM Press.

[74] Moni Naor and Benny Pinkas. Distributed oblivious transfer. In *Proc. ASIACRYPT 2000*, volume 1976, pages 200–219. Springer-Verlag, 2000. Lecture Notes in Computer Science, vol. 293.

[75] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 448–457, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[76] Moni Naor, Benny Pinkas, and Reuben Sumner. Privacy preserving auctions and mechanism design. In *1st ACM Conf. on Electronic Commerce*, pages 129–139, 1999.

[77] Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *ACM Conference on Computer and Communications Security*, pages 364–372, 2005.

[78] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. EUROCRYPT 99*, pages 223–238. Springer-Verlag, 1999. Lecture Notes in Computer Science, vol. 1592.

[79] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 245–254, New York, NY, USA, 2000. ACM Press.

[80] Benny Pinkas and Tomas Sander. Securing passwords against dictionary attacks. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 161–170, New York, NY, USA, 2002. ACM Press.

[81] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[82] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 433–444, London, UK, 1992. Springer-Verlag.

[83] R. L. Rivest, A. Shamir, and L. M. Adelman. A method for obtaining digital signatures and public-key cryptosystems. Technical Report LCS/TM-82, MIT, 1977.

[84] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1):96–99, 1983.

[85] Phillip Rogaway. *The round complexity of secure protocols*. PhD thesis, MIT, 1991.

[86] Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for NC^1. In *Proceedings 40th IEEE Symposium on Foundations of Computer Science*, pages 554–566, New York, 1999. IEEE.

[87] Victor Shoup. On formal models for secure key exchange. Technical Report RZ 3120 (#93166), IBM, 1999.

[88] Leslie G. Valiant. Universal circuits (preliminary report). In *STOC '76: Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 196–203, New York, NY, USA, 1976. ACM Press.

[89] A. C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 160–164, Chicago, 1982. IEEE.

[90] A. C. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symp. on Foundations of Comp. Science*, pages 162–167, Toronto, 1986. IEEE.