Cryptographic Reverse Firewalls

Ilya Mironov[1](✉) and Noah Stephens-Davidowitz[2]

[1] Google, Menlo Park, US
mironov@gmail.com
[2] Department of Computer Science, New York University, New York, US
noahsd@gmail.com

Abstract. Recent revelations by Edward Snowden [3,20,27] show that a user's own hardware and software can be used against her in various ways (e.g., to leak her private information). And, a series of recent announcements has shown that widespread implementations of cryptographic software often contain serious bugs that cripple security (e.g., [12–14,22]). This motivates us to consider the following (seemingly absurd) question: *How can we guarantee a user's security when she may be using a malfunctioning or arbitrarily compromised machine?* To that end, we introduce the notion of a *cryptographic reverse firewall* (RF). Such a machine sits between the user's computer and the outside world, potentially modifying the messages that she sends and receives as she engages in a cryptographic protocol.

A good reverse firewall accomplishes three things: (1) it *maintains functionality*, so that if the user's computer is working correctly, the RF will not break the functionality of the underlying protocol; (2) it *preserves security*, so that regardless of how the user's machine behaves, the presence of the RF will provide the same security guarantees as the properly implemented protocol; and (3) it *resists exfiltration*, so that regardless of how the user's machine behaves, the presence of the RF will prevent the machine from leaking any information to the outside world. Importantly, we do not model the firewall as a trusted party. It does not share any secrets with the user, and the protocol should be both secure and functional without the firewall (when the protocol's implementation is correct).

Our security definition for reverse firewalls depends on the security notion(s) of the underlying protocol. As such, our model generalizes much prior work (e.g., [5,7,26,32]) and provides a general framework for building cryptographic schemes that remain secure when run on compromised machine. It is also a modern take on a line of work that received considerable attention in the 80s and 90s (e.g., [7,9,11,15,16,30,31]).

We show that our definition is achievable by constructing a private function evaluation protocol with a secure reverse firewall for each party. Along the way, we design an oblivious transfer protocol that also has a secure RF for each party, and a rerandomizable garbled circuit that is both more efficient and more secure than previous constructions. Finally, we show how to convert *any* protocol into a protocol with an exfiltration-resistant reverse firewall for all parties. (In other words, we provide a generic way to prevent a tampered machine from leaking information to an eavesdropper via *any* protocol.)

Most of this work was done in Microsoft Research.

E. Oswald and M. Fischlin (Eds.): EUROCRYPT 2015, Part II, LNCS 9057, pp. 657–686, 2015.
DOI: 10.1007/978-3-662-46803-6_22

1 Introduction

Recent revelations of Edward Snowden show that powerful actors will go to remarkable lengths to obtain secret information. In particular, the National Security Agency has engineered a backdoor into a public cryptographic standard [3,27] and intercepted hardware as it was being delivered to customers in order to tamper with it [20]. Meanwhile, multiple serious flaws have been uncovered in widely used implementations of cryptographic protocols, leaving many users vulnerable to simple but devastating attacks (e.g., [12–14,22]). The extreme complexity of modern cryptographic implementations makes it extremely difficult for experts (let alone the typical user) to detect such vulnerabilities, even when they are introduced innocently. Attackers that deliberately insert such vulnerabilities into hardware and software can make this even harder by using cryptographic methods to cover their tracks.

So, facing the disturbing (and quite real) possibility of a compromise that reaches inside one's communication platform, we consider the following seemingly paradoxical question: *Can we design cryptographic protocols that achieve meaningful security when the adversary may arbitrarily tamper with the victim's computer?*

To resolve this question, we present a strong and general notion of security in the presence of an active tampering adversary and show how to instantiate powerful cryptographic primitives in this model. Of course, if Alice's computer simply chooses to replace her first message to Bob in some protocol with, for example, her secret business plans, we cannot hope to guarantee her security without some sort of help. Inverting the metaphor from network security, we propose and investigate the power of a (cryptographic) reverse firewall—an entity whose role is to protect cryptographic schemes and protocols from insider attacks. Informally, a cryptographic reverse firewall (RF) is a machine run by a third party (e.g., a security contractor hired by Alice's employer) that sits somewhere between Alice and the outside world and prevents Alice's computer from compromising her security by potentially modifying the messages that it sends and receives. In contrast to the standard firewall, the focus of a reverse firewall is on the inside of the perimeter. In particular, one important goal of reverse firewall is prevention of exfiltration attacks. Our primary contribution is the definition of reverse firewalls and the additional level of security that they bring to cryptographic protocols.

More specifically, we define three desirable properties of reverse firewalls. First, a reverse firewall should *maintain functionality*. I.e., if Alice's computer is behaving as it should, then the RF should not break the underlying functionality of the protocol. Second, a reverse firewall should *preserve security*. I.e., if the protocol without the RF present provides some security guarantee when Alice's computer behaves as it should, then the protocol *with* the RF present should provide this same security guarantee *regardless of how Alice's computer behaves*. Finally, a reverse firewall should *resist exfiltration*. Intuitively, an RF is exfiltration-resistant if Alice's tampered implementation cannot leak any information to the outside world through the firewall.

We defer much of the discussion of our definition to Section 2, where we introduce it formally. We emphasize, however, that the reverse firewall is *not* a trusted third party, and we do not rely solely on it for security. If Alice's implementation of the protocol is correct, then the protocol should be secure and functional without the firewall. In other words, we ask that the firewall *preserves* security, not that it *provides* it. In addition, the RF only has access to Alice's incoming and outgoing messages and any public parameters—not to Alice's state or input or any shared secrets. In effect, we place no more trust in the reverse firewall than we do in the communication medium. (We additionally require that firewalls be "stackable," so that one party may have arbitrarily many firewalls. Security is then guaranteed if just one of the firewalls is implemented correctly—or if Alice's own implementation is correct.)

Note that our security definition is quite strong, as it imagines the adversary "living inside of our computer." Consider, for example, a secure coin-flipping protocol in which Alice wishes to agree on a fair coin toss with Bob. Informally, the protocol is secure for Alice in the standard setting (i.e., without reverse firewalls) if Bob cannot bias the resulting coin toss alone. In our setting, we imagine both parties *working together* to bias the coin toss in Bob's favor. (Bob is adversarial as always, and in our setting, Bob may have also tampered with Alice's computer so that it is effectively "on Bob's side.") The only defense against this attack is a reverse firewall that can modify the messages that Alice sends and receives but must do so in a way that does not break the protocol when Alice and Bob are honest. (And, again, it must do so without access to any privileged information.)

In spite of this strength, we show that security in this model is achievable for very strong primitives. Indeed, we construct a two-round private function evaluation protocol that is secure in this model (Section 4). In particular, *each* party in this protocol has a corresponding secure reverse firewall. In other words, we show a relatively simple protocol that allows Alice and Bob to jointly and securely compute any circuit with the remarkable property that a reverse firewall can guarantee Alice's security even when Bob has tampered with her computer, and vice versa. This immediately shows that a very large class of two-party primitives can be realized securely in this model. The main ingredients for this protocol are an oblivious transfer scheme that itself has a secure reverse firewall for each party (Section 3) and a rerandomizable version of Yao's garbled circuit (Section 4.1). Our oblivious transfer protocol is a modified version of the Naor-Pinkas/Aiello-Ishai-Reingold protocol [1,25]. Our rerandomizable garbled circuit is significantly more efficient than the construction of Gentry et al. [19], and it achieves a stronger notion of rerandomizability. (See Section 1.1 for further comparison.)

Finally, in Section 5, we show a generic construction that can convert *any* protocol into a protocol with the same functionality that has an exfiltration-resistant reverse firewall. In other words, we provide a generic way to prevent a tampered machine from leaking information to an eavesdropper via any protocol. So, for the important special case in which Alice is primarily concerned

with passive eavesdroppers, we show that *any* multiparty functionality can be implemented in our model.

Our protocols are described in full in terms of basic group operations, and we avoid using "heavy machinery" like non-interactive zero-knowledge proofs in our constructions. In particular, this means that our protocols are relatively simple and efficient and that the security of our constructions follows from relatively weak complexity-theoretic assumptions (namely, the slight variants of the decisional Diffie-Hellman assumption presented in Appendix A).

1.1 Related Work

In this section we give a summary of related prior work, starting with the most directly comparable and recent literature. Given the size and the scope of existing work dealing with various models of insider attacks and mitigation strategies, our focus is on the similarities and differences between our work and prior art rather than a comprehensive review of all previous approaches.

Algorithm-substitution Attacks. Motivated by the potential threat of powerful adversaries subverting implementations of cryptographic algorithms, Bellare, Paterson, and Rogaway recently proposed a formalization of the notion of resilience of symmetric encryption schemes to algorithm-substitution attacks (ASA) [5]. They observe that modern standards for symmetric encryption crucially rely on sender-chosen randomness to attain acceptable security levels. Since these standards do not include any mechanisms for ensuring that randomness used in the encryption stage is unbiased, they effectively enable a communication channel, which a corrupt implementation may use to leak information to an external party.

Bellare et al. define a general framework for ASA security, identifying two adversarial goals—avoiding detection and conducting surveillance. They cast several algorithm-substitution attacks against symmetric-key encryption in this framework, showing that widely deployed secure communication protocols, such as SSL/TLS, IPsec, and SSH, are vulnerable to these attacks. Furthermore, they present a universal, essentially undetectable attack effective against any stateless, randomized symmetric-key encryption scheme.

On the positive (defensive) side, Bellare et al. advocate using stateful, deterministic encryption schemes with unique ciphertexts as a counter-ASA measure. They construct a provably ASA-resilient encryption scheme based on the encode-then-encipher paradigm, and prove that all nonce-based schemes satisfying a natural non-degeneracy condition can be converted into stateful schemes with unique ciphertexts by choosing their nonces sequentially.

Our work extends Bellare et al. in several directions. First, we include in our treatment arbitrary two- and multi-party protocols, as opposed to just symmetric-key encryption. Second, we shift our objective from developing primitives that are ASA-resilient by design to constructing protocols that are reverse-firewall-ready. Bellare et al. only achieve security against adversaries that do not break the functionality of the encryption scheme ("functionality-maintaining adversaries" in our

terminology). By making a stronger assumption—availability of an uncorrupted reverse firewall—we are able to achieve stronger security guarantees, such as security against tampered implementations that break functionality.

Our results and techniques can be viewed as complementary. Whereas Bellare et al. make a strong case for suppressing "freedom of choice" in cryptographic primitives, we demonstrate that additional randomness can be injected by an intermediary in some protocols to achieve stronger security guarantees for a much wider range of primitives.

Collusion-free Protocols and Mediated Collusion-free Protocols. Informally, Lepinski, Micali, and shelat say that a multi-party protocol is collusion-free if the parties cannot communicate information about their private inputs to each other via the protocol [23]. For example, a collusion-free protocol for the game of poker allows parties to play a hand of poker, but it does not allow them to communicate information about their cards to other players during the hand.

This notion resembles our definition of exfiltration resistance in that it disallows subliminal communication via the protocol, but the two notions are incomparable. On one hand, the definition of Lepinski et al. is much stronger than ours because it does not allow the use of a third-party reverse firewall to prevent subliminal communication. On the other hand, it is much weaker because it specifies what information parties are not allowed to communicate. Indeed, their constructions involve a setup phase that is conducted before the parties are given their inputs, and the authors observe that this setup phase can be used as a subliminal channel. So, in our model, their protocols are completely insecure. Their constructions also require strong physical assumptions to ensure verifiable determinism.

To avoid the need for the setup phase and physical assumptions, Alwen, shelat, and Visconti introduce the mediated model for collusion-free protocols [2]. In this model, all communication between the parties is routed through a mediator. Intuitively, the mediator rerandomizes the parties' messages in much the same way that our reverse firewalls do. However, the mediator is much more powerful than a reverse firewall in that (1) it intercepts all parties' messages and (2) it may exchange messages with the parties in any order. In contrast, our firewall modifies the messages sent and received by a single party in an online fashion, and we require our protocols to work without the firewall present. Because Alwen et al. give the mediator this additional power, they must explicitly model security against the mediator as a separate property of the protocol. In contrast, we get security "against the firewall" for free, as a natural consequence of the security of the underlying protocol. Their security definition is also stronger in the sense that it includes a strong notion of secure multi-party computation. While our notion of security preservation allows for such security, we intentionally do not require it in general.

Subliminal Channels and Divertible Protocols. A long series of works explored the idea of subliminal channels in various cryptosystems (e.g., [7,9,11,15,16,30,31]). Simmons [30] introduced the notion by showing subliminal channels in various signature and authentication schemes. The underlying theme of

this work is a story in which two prisoners, Alice and Bob, wish to communicate in some sanctioned way through the prison's warden (e.g., Alice wishes to tell Bob in some authenticated manner that she has not been harmed). The warden wishes to remove any subliminal messages from this communication (e.g., to prevent Alice from communicating escape plans to Bob). The warden in this story is quite similar to our reverse firewall, and the notion of a subliminal-free channel is closely related to our notion of exfiltration resistance. Because of the wide body of work with a variety of definitions, results, and applications, we focus on a small portion that is most related to our work—divertible protocols.

Intuitively, a protocol is divertible if a warden sitting between Alice and Bob can rerandomize the messages of both parties so that (1) neither party is aware of the warden's existence and (2) neither party can distinguish between an interaction with a dishonest party with the warden in the middle and an interaction with an honest party. Okamoto and Ohta provided the first definition of divertibility for zero-knowledge proofs [26] (based on earlier definitions of subliminal-free zero-knowledge proofs), and Burmester et al. showed that all languages in NP have a divertible zero-knowledge proof [9,10]. These simple and elegant constructions immediately provide zero-knowledge proofs with reverse firewalls for all languages in NP.

Blaze, Bleumer, and Strauss showed how to generalize and strengthen the definition of divertibility to apply to any two-party cryptographic protocol [7]. Indeed, their prescient definition comes close to our notion of a protocol with an exfiltration-resistant reverse firewall. We highlight three primary differences between their work and ours.

1. In our terminology, Blaze et al. consider only exfiltration resistance and not security preservation. In some applications (e.g., zero-knowledge proofs), the two properties are equivalent, but in many important applications (such as those that we consider in the sequel), the two properties are very different. (See Section 2.3 for further discussion of the distinction between these two properties.)
2. Blaze et al. implicitly assume that any dishonest version of the prover still provides valid proofs. (In our language, tampered provers must "maintain functionality.") This assumption is necessary for the prover of zero-knowledge proofs, but in general we can and should do better.
3. They consider only synchronous protocols with two parties and one warden. We consider asynchronous multi-party protocols in which each party may have its own firewall. By separating the warden into multiple firewalls and moving away from the synchronous model, our definition becomes much stronger, as our firewalls do not have the benefit of seeing all messages from all parties sent during a round before deciding how to modify them. (Indeed, Blaze et al. provide an example of a simple divertible key-agreement protocol. However, this protocol is *not* secure in our model because it crucially relies on the synchronous model of communication for its security.) We also find our more modular model to be more natural in our modern context, in which different parties may have different security needs.

Divertible protocols also differ from protocols with reverse firewalls in a number of more subtle ways. For example, Blaze et al. require that the warden is undetectable to either party. The protocols presented in the sequel achieve this notion of "transparency", but we intentionally do not require it as part of our definition.

In short, divertible protocols and subliminal-free channels were founded on a story that predates the concerns that motivate our work. Our more modern story, in which Alice and Bob (who need not be prisoners!) are concerned that their computers have been corrupted, leads naturally to our more general definition.

Kleptography. Young and Yung identified an important subclass of insider threats against cryptographic schemes, which they called *kleptographic attacks* [32]. The goal of a kleptographic attack is to leak a secret to an adversary who planted a malicious implementation of a cryptographic system on a victim's computer. The attack is asymmetric—the compromised implementation may carry the attacker's public key, but a private key is necessary in order to read from the subliminal channel. A secure kleptographic attack is undetectable as long as the system is accessed as a black box, and while it may be identified if one reverse engineers the implementation, this will only expose the attacker's public key. In particular, if multiple systems run the same compromised software stack, a successful reverse engineering effort of one such system will not help in breaching the security of others.

The (now withdrawn) NIST-standardized Dual Elliptic Curve Determnistic Random Bit Generator (Dual_EC_DRBG) is an example of a mechanism with a potential kleptographic backdoor [8,29].

Our adversarial model is a relaxation of the kleptographic attacker. We consider the possibility that the adversary may not worry about detection and is not concerned about a split-key solution.

Rerandomizable Garbled Circuits. Gentry, Halevi, and Vaikuntanathan construct a rerandomizable version of Yao's garbled circuit in order to build an "i-hop" homomorphic encryption scheme [19]. Their construction is quite elegant, and its security is based on a slightly weaker assumption than ours (pure decisional Diffie-Hellman, as opposed to the slight variants presented in Appendix A). But, it does not work in our context. Informally, their circuit is rerandomizable when constructed honestly, but the rerandomization of a dishonestly constructed circuit can easily be distinguished from a freshly garbled circuit. (With negligible probability, even the honest implementation can create circuits that in some sense "cannot be rerandomized.") In our context, in which we consider the possibility that the garbled circuit was constructed by a corrupted algorithm, this is a fatal flaw. We thus construct a new garbling scheme that can be rerandomized in a much stronger sense.

Our scheme (presented in Section 4) is also substantially more efficient than that of Gentry et al. The size of a single gate in their circuit is $O(\lambda^2)$ group elements, where λ is the security parameter, whereas our gates require only a constant number of group elements. As a consequence, our rerandomizable garbling scheme (which uses a trick inspired by Prabhakaran and Rosulek [28])

also implies a significantly more efficient implementation of i-hop homomorphic encryption.

Combiners. An alternative defense against untrusted implementations of a cryptographic primitive is to *combine* multiple implementations of the same primitive in some way so that the combined primitive will be secure if a suitably large subset of the initial primitives are secure. This idea is quite common in the literature, and it was formalized by Harnik et al., who show that many primitives have elegant robust combiners [21].

Combiners solve a slightly different problem than reverse firewalls. Firewalls guarantee security when a user's system has been arbitrarily compromised, while combiners provide security only when the user already has access to at least one secure implementation of a primitive (and a secure implementation of the combiner itself!). Intuitively, combiners are applicable when multiple implementations of the same primitive exist that either (1) may have bugs in them or (2) rely on different unproven assumptions. In contrast, reverse firewalls work even when our implementations have been intentionally compromised.

2 Cryptographic Reverse Firewalls

We now present our general definition of a cryptographic reverse firewall that can be applied to a large class of primitives. This requires us first to define a cryptographic protocol in a (very general) way that suits our purposes. We note, however, that we describe the concrete schemes presented in the sequel in simpler terms. So, this level of generality is not necessary to understand the rest of the paper.

2.1 Cryptographic protocols

Definition 1 (Cryptographic protocol). *A* cryptographic protocol $\mathcal{P}$ *defines an interaction between stateful* parties $(P_1, \ldots, P_\ell)$*. First, a setup procedure* $\mathsf{setup}(1^\lambda)$ *is run, where* λ *is the security parameter. It returns a starting state for each party* $(\sigma_{P_i})_{i=1}^{\ell}$*, which we call their respective* input*; public parameters* ρ*; and a schedule of messages.*[1] *The parties proceed to send messages to each other according to the schedule. Each party has an associated* next message algorithm $\mathsf{next}_{P_i}(\sigma_{P_i})$ *that is called when it must output a message and a* message receipt algorithm $\mathsf{receive}_{P_i}(\sigma_{P_i}, m)$ *that is called upon receipt of a message to update the party's state. After the protocol is finished, each party runs its* output algorithm $\mathsf{output}_{P_i}(\sigma_{P_i})$ *and returns the result.*

[1] Note that we only consider protocols in which the message schedule is fixed by setup. Formally, this schedule determines the number of messages that a party must receive from each other party before sending each message. (E.g., Alice will send her second message after she has received three messages from Bob, two from Carol, and one from David.) We omit explicit reference to this schedule in the sequel as it will always be clear.

We identify the protocol with its parties and setup procedure, $\mathcal{P} = (\mathsf{setup}, (P_i)_{i=1}^{\ell})$*, and we identify the parties with the algorithms that define them,* $P_i = (\mathsf{receive}_{P_i}, \mathsf{next}_{P_i}, \mathsf{output}_{P_i})$*. A complete record of all messages sent during a run of the protocol is a* transcript $\mathcal{T}$*.*

We call a run of a protocol a run with input I *if the parties' respective input and the public parameters are set to the values represented by* I*. We assume implicitly that the input* I *satisfies certain* validity requirements*.*

A protocol must satisfy functionality requirements $\mathcal{F}$*, which place constraints on the output of the parties for particular input* I*, and* security requirements $\mathcal{S}$*, which place constraints on the message distribution conditioned on specific input* I*. For our purposes, it will often be convenient to assign to each security requirement* $\mathcal{S}$ *a specific party who "is concerned with* $\mathcal{S}$*." For a party* P*, we say that a protocol is* secure for P *if all of* P*'s security requirements are met.*

For example, a one-out-of-two oblivious transfer (OT) protocol is a protocol between a sender, Alice, and a receiver, Bob. Alice's input is a pair of messages m_0, m_1, and Bob's input is a bit b. The protocol is functional if Bob's output is m_b. It is secure for Bob if for any valid messages m_0, m_1, no efficient algorithm playing the role of Alice can predict b with non-negligible advantage after the protocol is complete. (I.e., Alice is *oblivious* to Bob's bit b.) Intuitively, the protocol is secure for Alice if no efficient algorithm playing the role of Bob can "learn any information" other than m_0 or m_1 (but not both!).

Below, we list some terminology and notation that will be useful in the next section.

Definition 2 (Protocols and parties). *For a protocol* $\mathcal{P} = (\mathsf{setup}, (P_i)_{i=1}^{\ell})$ *satisfying functionality* $\mathcal{F}$*, input* I*, party* P*, index* j*, and index set* $J \subseteq \{1, \ldots, \ell\}$*,*

1. $\mathcal{T} \leftarrow \mathcal{P}(I)$ *denotes setting the variable* $\mathcal{T}$ *to the transcript obtained by a run of* $\mathcal{P}$ *with input* I*;*
2. $\mathcal{P}_{P_j \Rightarrow P}$ *is the protocol obtained by replacing party* P_j *with* P *in the protocol* $\mathcal{P}$*;*
3. $\mathcal{P}_{J \Rightarrow P}$ *is the protocol obtained by replacing all of the parties* $\{P_j\}_{j \in J}$ *with a single implementation of* P *in* $\mathcal{P}$ *(i.e., the parties* $\{P_j\}_{j \in J}$ *"collapse" into a single party* P *that has a single state* σ_P*);*
4. *if any party sends the special symbol* $\perp$ *as a message at any time, then the protocol immediately ends and, by definition, functionality has been violated; and*
5. P maintains $\mathcal{F}$ for P_j in $\mathcal{P}$ *if* $\mathcal{P}_{P_j \Rightarrow P}$ *satisfies* $\mathcal{F}$ *with all but negligible probability over the random coins of the parties and* setup *procedure of* P *for any fixed input.*

When $\mathcal{F}$*,* P_j*, and* $\mathcal{P}$ *are clear, we simply say that* P maintains functionality*.*

2.2 Cryptographic Reverse Firewalls

Definition 3 (Cryptographic reverse firewall). *A* cryptographic reverse firewall *(RF) is a stateful algorithm* $\mathcal{W}$ *that takes as input its state and a*

message and outputs an updated state and message. For simplicity, we do not write the state of $\mathcal{W}$ explicitly.

For a party $P = (\mathsf{receive}, \mathsf{next}, \mathsf{output})$ and reverse firewall $\mathcal{W}$, the composed party *is defined as*

$$\begin{aligned}\mathcal{W} \circ P := (&\mathsf{receive}_{\mathcal{W}\circ P}(\sigma, m) = \mathsf{receive}_P(\sigma, \mathcal{W}(m)),\\ &\mathsf{next}_{\mathcal{W}\circ P}(\sigma) = \mathcal{W}(\mathsf{next}_P(\sigma)),\\ &\mathsf{output}_{\mathcal{W}\circ P}(\sigma) = \mathsf{output}_P(\sigma))\ .\end{aligned}$$

When the composed party engages in a protocol, the state of $\mathcal{W}$ is initialized to the public parameters ρ. If $\mathcal{W}$ is meant to be composed with a party P, we call it a reverse firewall for P.

Intuitively, an RF simply modifies Alice's incoming and outgoing messages. Alice of course does not want a reverse firewall to ruin her protocol's functionality when her internal implementation is correct. Indeed, we want something more than this. Alice's employer may wish to deploy multiple reverse firewalls (one internal firewall managed by its network administrators, one provided by a security contractor, another by networking equipment vendor, etc.), and we do not want such "stacking" of firewalls to break functionality. The definition below captures this.

Definition 4 (Functionality-maintaining RFs). *For any reverse firewall $\mathcal{W}$ and any party P, let $\mathcal{W}^1 \circ P = \mathcal{W} \circ P$, and for $k \geq 2$, let $\mathcal{W}^k \circ P = \mathcal{W} \circ (\mathcal{W}^{k-1} \circ P)$.*

For a protocol $\mathcal{P}$ that satisfies some functionality requirements $\mathcal{F}$, we say that a reverse firewall $\mathcal{W}$ maintains *$\mathcal{F}$ for P_j in $\mathcal{P}$ if $\mathcal{W}^k \circ P_j$ maintains $\mathcal{F}$ for P_j in $\mathcal{P}$ for any polynomially bounded $k \geq 1$. When $\mathcal{F}$, P_j, and $\mathcal{P}$ are clear, we simply say that $\mathcal{W}$* maintains functionality.

We emphasize that we are interested in reverse firewalls that *maintain* the functionality of an already functional protocol—protocols that do not function without the firewall are not nearly as interesting. We also note that the reverse firewalls described in the sequel actually achieve much stronger properties. In particular, they are all "transparent", so that the behavior of $\mathcal{W} \circ P$ is identical to the behavior of P if P is the honest implementation. And, $\mathcal{W} \circ P$ is functionality maintaining whenever P is (and not just when P is an honest implementation). While these properties seem desirable for many applications, we do not wish to exclude from our definitions firewalls that, for example, append a signature to each message that they send.

More interestingly, we would like a reverse firewall to protect Alice from an adversary that may have tampered with her computer. To that end, we ask that the firewall *preserves* the security properties of the underlying protocol. So, we are only interested in protocols that are already secure without the firewall present. Since this definition depends on the security properties of the underlying protocol, it provides a general framework for the study of arbitrary cryptographic primitives in this model. Our strongest notion of security imagines a

completely adversarial algorithm replacing Alice's implementation of the protocol and requires that security is still preserved even in this setting. Our weaker notion only considers tampered implementations that maintain functionality.

Definition 5 (Security-preserving RFs). *For a protocol* $\mathcal{P} = (\mathsf{setup}, (\mathsf{next}_{P_i}, \mathsf{receive}_{P_i}, \mathsf{output}_{P_i})_{i=1}^{\ell})$ *that satisfies some security requirements* $\mathcal{S}$ *and functionality* $\mathcal{F}$ *and a reverse firewall* $\mathcal{W}$*,*

1. $\mathcal{W}$ strongly preserves $\mathcal{S}$ for P_j in $\mathcal{P}$ *if the protocol* $\mathcal{P}_{P_j \Rightarrow \mathcal{W} \circ P^*_{\mathcal{A}}}$ *satisfies* $\mathcal{S}$ *for any probabilistic polynomial-time* $P^*_{\mathcal{A}}$*; and*
2. $\mathcal{W}$ weakly preserves $\mathcal{S}$ for P_j in $\mathcal{P}$ against $\mathcal{F}$-maintaining adversaries *if the protocol* $\mathcal{P}_{P_j \Rightarrow \mathcal{W} \circ P^*_{\mathcal{A}}}$ *satisfies* $\mathcal{S}$ *for any probabilistic polynomial-time* $P^*_{\mathcal{A}}$ *that maintains functionality* $\mathcal{F}$*.*

When $\mathcal{S}$*,* P_j*,* $\mathcal{P}$ *and* $\mathcal{F}$ *are clear, we simply say that* $\mathcal{W}$ strongly preserves security *or* weakly preserves security *respectively.*

One type of attack that particularly concerns us is *exfiltration*, in which Alice's corrupted computer attempts to leak some private information (e.g., secret business plans) to an adversary who has control over some (possibly empty) list of other parties J. We call security against such an attack *exfiltration resistance*, and we define it in terms of the game $\mathsf{LEAK}(\mathcal{P}, P_j, J, \lambda)$, presented in Figure 1. Intuitively, the game LEAK asks the adversary to distinguish between a tampered implementation of party P_j and an honest implementation. An exfiltration-resistant reverse firewall therefore prevents an adversary from even learning whether Alice's computer has been compromised—let alone her secret business plans.

$$
\begin{array}{l}
\textbf{proc. } \mathsf{LEAK}(\mathcal{P}, P_j, J, \mathcal{W}, \lambda) \\
(\sigma_{\mathcal{A}}, P^*_{\mathcal{A}}, P^*_{\mathcal{B}}, I) \leftarrow \mathcal{A}(1^{\lambda}) \\
b \xleftarrow{\$} \{0,1\} \\
\texttt{IF } b = 1,\ P^* \leftarrow \mathcal{W} \circ P^*_{\mathcal{A}} \\
\texttt{ELSE},\ P^* \leftarrow \mathcal{W} \circ P_j \\
\mathcal{T}^* \leftarrow \mathcal{P}_{P_j \Rightarrow P^*, J \Rightarrow P_{\mathcal{B}^*}}(I) \\
b^* \leftarrow \mathcal{A}(\sigma_{\mathcal{A}}, \mathcal{T}^*, \sigma_{P^*_{\mathcal{B}}}) \\
\texttt{OUTPUT } (b = b^*)
\end{array}
$$

Fig. 1. $\mathsf{LEAK}(\mathcal{P}, P_j, J, \lambda)$, the exfiltration resistance security game for a reverse firewall $\mathcal{W}$ for party P_j in protocol $\mathcal{P}$ with corrupted parties J and security parameter λ. (Formally, the adversary represents a party by a collection of three (possibly randomized) circuits that implement the relevant functions $\mathsf{receive}$, next, and output.)

Definition 6 (Exfiltration-resistant RFs). *For a protocol* $\mathcal{P}$ *satisfying functionality* $\mathcal{F}$ *and a reverse firewall* $\mathcal{W}$*,*

1. $\mathcal{W}$ *is* $(\mathcal{P}, P_j, J)$-strongly exfiltration-resistant *if no PPT adversary* $\mathcal{A}$ *achieves advantage that is non-negligible in the security parameter* λ *in the game* $\mathsf{LEAK}(\mathcal{P}, P_j, J, \mathcal{W}, \lambda)$*; and*

2. $\mathcal{W}$ *is* $(\mathcal{P}, P_j, J)$-weakly exfiltration-resistant against $\mathcal{F}$-maintaining adversaries *if no PPT adversary* $\mathcal{A}$ *achieves advantage that is non-negligible in the security parameter* λ *in the game* $\mathsf{LEAK}(\mathcal{P}, P_j, J, \mathcal{W}, \lambda)$ *provided that the adversary's output* $P^*_{\mathcal{A}}$ *maintains* $\mathcal{F}$ *for* P_j.

When P_j, $\mathcal{P}$, *and* $\mathcal{F}$ *are clear, we simply say that* $\mathcal{W}$ *is* strongly exfiltration-resistant against J *or* weakly exfiltration-resistant against J *respectively. In the special case when* J *is empty, we say that* $\mathcal{W}$ *is* exfiltration-resistant against eavesdroppers.

This brings us to our strongest security notion.

Definition 7 (Robust RFs). *A cryptographic reverse firewall* $\mathcal{W}$ *is* robust for a party P_j in $\mathcal{P}$ with functionality requirements $\mathcal{F}$ and security requirements $\mathcal{S}$ *if it is* $\mathcal{F}$*-maintaining, strongly* $\mathcal{S}$*-preserving, and strongly exfiltration-resistant against the collection of all parties other than* P_j *in* $\mathcal{P}$. *We often simply say that* $\mathcal{W}$ *is robust when* P_j, $\mathcal{P}$, $\mathcal{F}$, *and* $\mathcal{S}$ *are clear.*

2.3 Discussion of the Definitions

The newly introduced terminology for security of reverse firewalls and a new type of adversary facilitates accurate and tight characterization of security guarantees offered by schemes in the sequel. Before we proceed to constructions, some analysis of the new notions is in order.

The Relationship Between Exfiltration-resistant and Security-preserving Firewalls. For many natural notions of security, exfiltration resistance and security preservation are equivalent. For example, a reverse firewall preserves the semantic security of an encryption scheme if and only if it is exfiltration-resistant against an eavesdropper. However, for notions of security that do not promise privacy, a security-preserving firewall is not necessarily exfiltration-resistant. For example, a reverse firewall may preserve the binding property of a commitment scheme, but it may still allow information to leak out of a compromised machine. Even when a security requirement does imply some type of privacy, a firewall that preserves it may not be exfiltration-resistant. Consider the hiding property of a commitment scheme, which guarantees privacy during the initial (commitment) phase, but it certainly does not prevent information from leaking during the opening phase. In fact, it is relatively easy to construct reverse firewalls that strongly preserve the hiding property of commitment schemes (just use a rerandomizable commitment scheme), but it is provably impossible to construct a strongly exfiltration-resistant reverse firewall for the sender against the receiver in any commitment scheme! (Loosely speaking, the functionality of a commitment scheme allows the sender to communicate a message to the receiver. So, a reverse firewall cannot hope to simultaneously maintain functionality and prevent the sender from leaking information to the receiver.)

On the other hand, it may seem at first that an exfiltration-resistant reverse firewall always preserves security, since interaction with such an RF composed

with an adversarially chosen circuit is, by definition, indistinguishable from interaction with an honest implementation. (Technically, we ask that the RF composed with an adversarially chosen circuit is indistinguishable from *the RF composed with* an honest implementation.) However, this is not always the case. For example, if security requirements are simulation-based or consider adversaries who have access to oracles or are computationally unbounded, then an exfiltration-resistant firewall may not preserve security.

Functionality-maintaining Adversaries. Intuitively, our weaker security notions exclude the "more conspicuous" adversaries whose tampered circuit would be noticed by honest parties participating in the protocol with non-negligible probability. However, even our weakest adversaries may behave arbitrarily some negligible fraction of the time against honest parties. This distinction can be quite important in the context of security definitions that allow for the corruption of other players in the protocol. For example, consider an oblivious transfer protocol in which Bob's first message is uniformly random over some large set (as is the case in Section 3). A tampered implementation of Alice in this protocol may respond to one specific such message by, say, encoding the XOR of both of Alice's inputs into its response to Bob. Such an implementation can still be functionality-maintaining because this event rarely happens when Bob behaves honestly. But, the security definition of oblivious transfer requires that an adversary playing the role of Bob should not be able to learn the XOR of the inputs.

So, any reverse firewall that even *weakly* preserves Alice's security in such a model must somehow address this issue. In Section 3, we approach it by composing a firewall for Alice that only works against tampered implementations that *always* maintain functionality with a firewall that is exfiltration-resistant *for Bob against Alice*. (The composed firewall is still serving Alice's security—Bob's outputs are randomized for her protection, not his.) We expect this approach to be useful in future work.

Timing and Scheduling Issues. Our model does not explicitly account for the timing of messages. In practice, message timing is a natural channel, and a tampered implementation could, of course, use this to leak information and compromise Alice's security. So, any reverse firewall in the real world must account for this (e.g., by fixing the time between when it receives a message and when it forwards Alice's response). As the above discussion shows, in some cases, it might be necessary for the firewall to control the timing of both outgoing *and* incoming messages. In a protocol with more than two parties, this issue naturally becomes more complicated. In such cases, protocol designers should consider the relative timing of messages from multiple parties' perspectives and the order in which Alice receives messages from various parties.

3 Oblivious Transfer

Naor and Pinkas and, independently, Aiello, Ishai, and Reingold developed very similar OT protocols whose security reduces immediately to DDH [1,25]. We

present a version of this protocol that is suitable for our setting. In particular, Alice's input is a pair of elements (m_0, m_1) in some group G of order p in which DDH is hard, and Bob's input is a bit b. Alice and Bob then engage in the protocol shown in Figure 2.

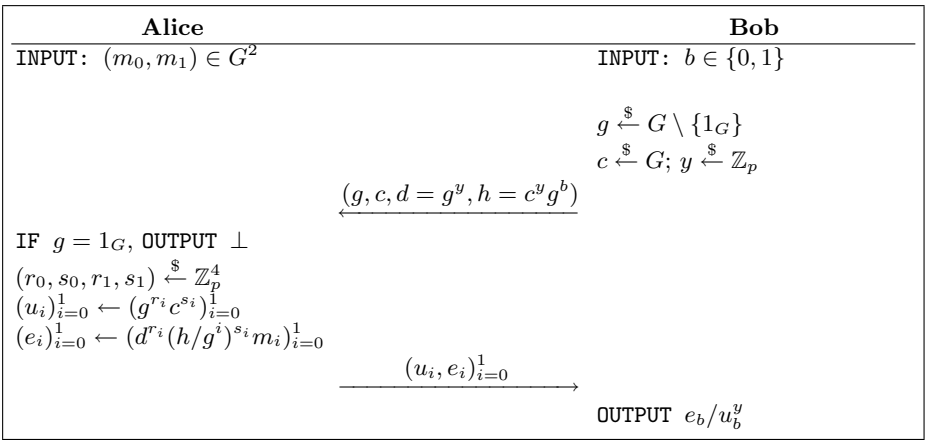

Fig. 2. A version of Naor-Pinkas/Aiello-Ishai-Reingold protocol for oblivious transfer

Proposition 1. *The protocol shown in Figure 2 is correct and secure for both parties if DDH is hard in G.*

Proof. Let $x = \log_g c$, which is well defined since $g \neq 1_G$ and G is cyclic. Correctness follows from the fact that $u_b^y = g^{s_b xy + r_b y} = (h/g^b)^{s_b} d^{r_b}$. Bob's security follows immediately from the DDH assumption in G.

To prove security for Alice, we note that if $(g, c, d, h) \neq (g, g^x, g^y, g^{xy+b})$ for some $x, y \in \mathbb{Z}_p$ and $b \in \{0, 1\}$, then (u_b, e_b) is uniformly random. Indeed, note that

$$\log_g u_b = r_b + x \cdot s_b,$$
$$\log_g(e_b/m_b) = y \cdot r_b + (\log_g h - b) \cdot s_b.$$

It follows that u_b and e_b are distributed uniformly and independently unless $\log_g h - b = xy$. This allows us to construct, for any (not necessarily efficient) adversary $\mathcal{B}$ playing the role of Bob, an (inefficient) simulator $S_{\mathcal{B}}$ with access to the ideal functionality $\mathcal{F}$ that behaves as follows on input b.

1. $(\sigma, g, c, d, h) \leftarrow \mathcal{B}()$.
2. $(m_0, m_1) \xleftarrow{\$} G^2$.
3. If $(g, c, d, h) = (g, g^x, g^y, g^{xy+b})$ for $b \in \{0, 1\}$, set $m_b \leftarrow \mathcal{F}(b)$.
4. $(r_0, r_1, s_0, s_1) \xleftarrow{\$} \mathbb{Z}_p$.

5. $(u_i, e_i)_{i=0}^1 \leftarrow (g^{r_i} c^{s_i}, d^{r_i} (h/g^i)^{s_i} m_i)_{i=0}^1$.
6. Output $\mathcal{B}(\sigma, u_0, e_0, u_1, e_1)$.

It should be clear that the simulator's "message" $(u_i, e_i)_{i=0}^1$ is distributed identically to the message that $\mathcal{B}$ receives from Alice in the real protocol, and the result follows. □

We present reverse firewalls for both parties in our variant of the Naor-Pinkas/Aiello-Ishai-Reingold protocol and show that they are secure. Bob's reverse firewall is shown in Figure 3, and Alice's is shown in Figure 4. Alice's firewall by itself strongly prevents leaks against eavesdroppers. In order for it to weakly maintain security, it must be composed with Bob's firewall.

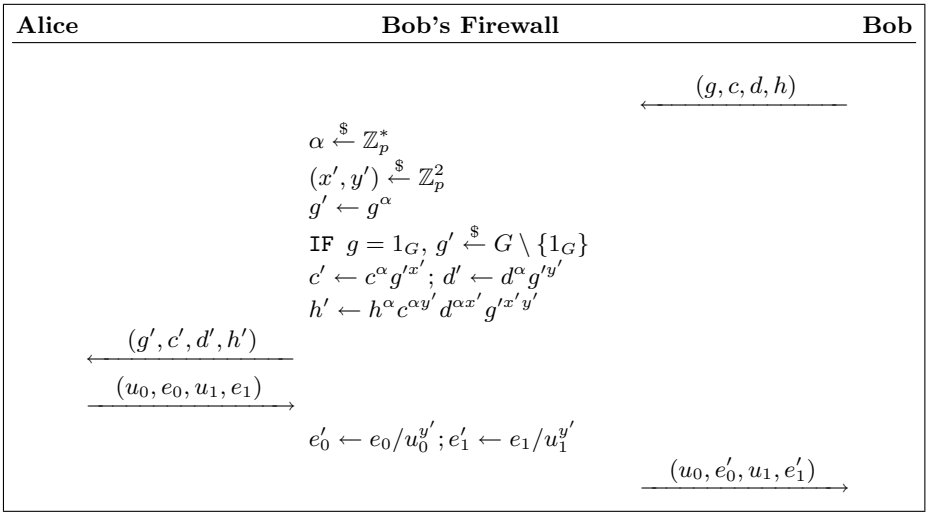

Fig. 3. Bob's reverse firewall for the protocol shown in Figure 2

In the full version of the paper [24], we prove the following theorem.

Theorem 2. *Bob's reverse firewall $\mathcal{W}_\mathsf{B}$ shown in Figure 3 maintains correctness and is robust if the chosen-base DDH with a hint game is hard in G.*

Alice's reverse firewall $\mathcal{W}_\mathsf{A}$ shown in Figure 4 maintains correctness and is strongly exfiltration-resistant against an eavesdropper if DDH is hard in G. The composed firewall $\mathcal{W}_\mathsf{B} \circ \mathcal{W}_\mathsf{A}$ also weakly preserves security against Bob.

4 Private Function Evaluation

We now construct a private function evaluation scheme based on the oblivious transfer protocol from Section 3 and a version of Yao's garbled circuit. We assume that the reader is familiar with garbled circuits and this type of construction (see, for example, [4]). Our key technical tool is a rerandomizable garbled circuit based on ElGamal encryption [17], which may be of independent interest.

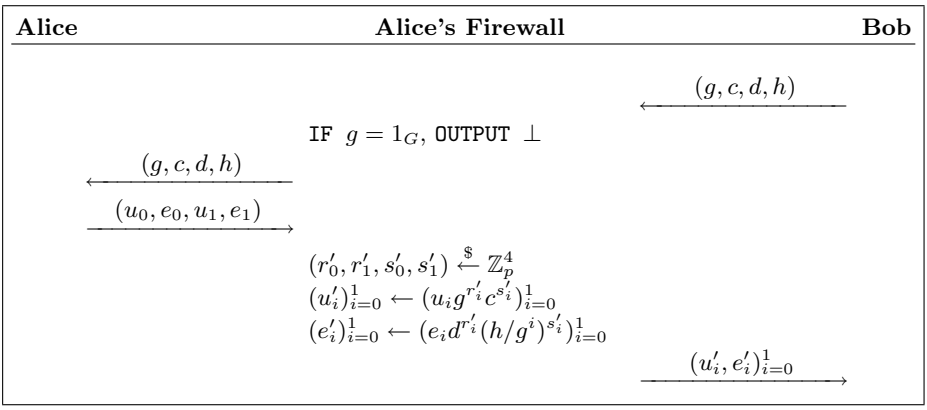

Fig. 4. Alice's reverse firewall for the protocol shown in Figure 2. It is strongly exfiltration-resistant, and it weakly preserves security when it is composed with the firewall shown in Figure 3.

4.1 A Rerandomizable Garbled Circuit

We wish to use the homomorphic properties and rerandomizability of ElGamal encryption to make a rerandomizable garbled circuit. But, a subtlety immediately arises: Yao's garbled circuit construction makes heavy use of encryptions of private keys (which can be used to decrypt more encryptions of private keys, etc.). However, in ElGamal encryption, private keys are elements in $\mathbb{Z}_p$ but messages are elements of a group G of order p in which DDH is hard. Our construction requires an efficient injective homomorphism from the key space to the message space. But, since DDH is easy in $\mathbb{Z}_p$, such a map cannot exist.

To get around this issue, we use a technique inspired by Prabhakaran and Rosulek [28]. In particular, for circuits of depth D, we need groups $G_1, \ldots, G_D$ of prime order $|G_d| = p_d$ such that G_d is a subgroup of $\mathbb{Z}^*_{p_{d+1}}$ and p_{d+1}/p_d is polynomially bounded[2] for $d < D$. In particular, this means that, given $g \in G_d$ and $h \in G_{d+1}$, the operation h^g is well-defined, and elements from G_d can therefore serve as private keys for ElGamal encryption over G_{d+1}.

Formally we say that a vertex z is at depth d in a circuit layout $\mathcal{L}$ if the longest path from an input vertex to z has length $d-1$, and we write $\mathsf{depth}(z) = d$. For ease of presentation, we assume that all edges in the circuit layout $\mathcal{L}$ are between vertices of adjacent depths (i.e., edges do not "skip levels") and that all output vertices have maximal depth D. With this simplification, we can use the group G_d to garble vertices at depth d. (Note that this restriction is not necessary, and the garbling scheme generalizes naturally to handle arbitrary circuits.)

[2] In practice, such chains of primes can be found efficiently. Indeed, the sequence defined by starting at $q_1 = 2$ and setting q_{i+1} to be the minimal prime with $q_{i+1} \equiv 1 \mod q_i$ is suitable. There are 497 primes in this sequence between 2^{1024} and 2^{6144}. The ratios between successive primes in this specific chain are conjectured to remain polynomially bounded in the length of the primes, and other chains with this property are conjectured to be abundant. See, for example, [18].

Our garbling scheme for a circuit $\mathcal{C}$ is shown in Figure 5, and a schematic illustration of gate evaluation is provided in Figure 6. Alice can use the function Garble to garble a circuit $\mathcal{C}$ (represented by a collection of gate functions (f_z)), yielding a collection of ciphertexts (A_z) and *input tags* $(T_z^{(b)})_{z\in\mathcal{I},b\in\{0,1\}}$. Given a collection of ciphertexts (A_z) and input tags $(T_z^{(x_z)})_{z\in\mathcal{I}}$ corresponding to some input x, Bob can use the function Eval to compute $\mathcal{C}(x)$.

In particular, Garble assigns two tags $T_z^{(0)}$ and $T_z^{(1)}$ to each vertex z, which represent the vertex taking the value 0 and 1 respectively. $T_z^{(b)}$ is a uniformly random group element for all vertices that are not output gates, while the tags of output gates are simply encodings g_D^b of output bits. Intuitively, we want Bob to "only be able to learn" the tag corresponding to the value that each gate takes when $\mathcal{C}$ is evaluated on his input.

The function Garble represents each non-input gate z by A_z, a collection of ElGamal ciphertexts. The ciphertexts are encrypted under one of four private keys, each of which is the *product* of a tag from the gate's left parent $T_{L(z)}^{(b_L)}$ and a tag from the gate's right parent $T_{R(z)}^{(b_R)}$. The ciphertexts contain encryptions under the private key $T_{L(z)}^{(b_L)} \cdot T_{R(z)}^{(b_R)}$ of the tag $T_z^{f_z(b_L,b_R)}$ corresponding to the gate's output on some input (b_L, b_R). The ciphertexts are arranged randomly in the collection so that their order does not reveal any information about the circuit. So that Bob can know *which* ciphertext he should decrypt at each gate, together with each encrypted tag we also include a second ciphertext that encrypts a location bit τ (under the same key). Bob can then use the two location bits from a gate's left and right parent to know which ciphertext C_{τ_L,τ_R} to decrypt at the current gate.

The output of Garble is a collection of tags and location bits $(T_z^{(b)}, \tau_z^{(b)})_{z\in\mathcal{I},b\in\{0,1\}}$ for the input vertices together with the ciphertexts $(A_z)_{z\in V\setminus\mathcal{I}}$. The function Eval takes as input the ciphertexts (A_z) and the tags and location bits corresponding to some input x, $(T_z^{(x_z)}, \tau_z^{(x_z)})$, and it outputs $\mathcal{C}(x)$.

4.2 PFE from Garbled Circuits and OT

With the garbling scheme from Figure 5 and the oblivious transfer scheme from Section 3, we can build a private function evaluation protocol, which we present in Figure 7. We note that the protocol can be optimized so that Bob sends his oblivious transfer messages in one batch. With this optimization, the protocol requires only two messages. (Bob sends (g_1, c) and his oblivious transfer requests in a single message. Alice then sends her responses to the oblivious transfer requests and the garbled circuit, also in a single message.) The proof of security as well as the reverse firewalls and their corresponding proofs of security can be naturally modified to accommodate this change.

We prove the following proposition in the full version of the paper.

proc. $\mathsf{Garble}(\mathcal{C} = (f_z), g_1)$
FOR $d = 2, \ldots, D$,
  $g_d \xleftarrow{\$} G_d \setminus \{1_{G_d}\}$
FOR z in V,
  $b_z^* \xleftarrow{\$} \{0,1\}$
  IF $z \in \mathcal{O}$,
    $(T_z^{(0)}, T_z^{(1)}) \leftarrow (1_{G_D}, g_D)$
  ELSE,
    $d \leftarrow \mathsf{depth}(z)$
    $(T_z^{(0)}, T_z^{(1)}) \xleftarrow{\$} G_d^2$
FOR z in $V \setminus \mathcal{I}$,
  $A_z \xleftarrow{\$} \mathsf{GarbleGate}(z)$
FOR z in $\mathcal{I}$,
  $\tau_z^{(0)} \leftarrow b_z^*; \tau_z^{(1)} \leftarrow 1 \oplus b_z^*$
OUTPUT $((T_z^{(b)}, \tau_z^{(b)}), (A_z), (g_d))$

proc. $\mathsf{GarbleGate}(z)$
$d \leftarrow \mathsf{depth}(z)$
FOR (b_L, b_R) in $\{0,1\}^2$,
  $k \leftarrow T_{L(z)}^{(b_L)} \cdot T_{R(z)}^{(b_R)}$
  $\tau_L = b_L \oplus b_{L(z)}^*$
  $\tau_R = b_R \oplus b_{R(z)}^*$
  $\boldsymbol{\eta} \leftarrow (\tau_L, \tau_R)$
  $h_{\boldsymbol{\eta}} \leftarrow g_d^k$
  $b \leftarrow f_z(b_L, b_R)$
  $(r, s) \xleftarrow{\$} \mathbb{Z}_{p_d}^2$
  $(u_{\boldsymbol{\eta}}, e_{\boldsymbol{\eta}}) \leftarrow (g_d^r, h_{\boldsymbol{\eta}}^r T_z^{(b)})$
  $\tau \leftarrow b \oplus b_z^*$
  $(v_{\boldsymbol{\eta}}, w_{\boldsymbol{\eta}}) \leftarrow (g_d^s, h_{\boldsymbol{\eta}}^s g_d^\tau)$
OUTPUT $(h_{\boldsymbol{\eta}}, u_{\boldsymbol{\eta}}, e_{\boldsymbol{\eta}}, v_{\boldsymbol{\eta}}, w_{\boldsymbol{\eta}})$

proc. $\mathsf{Eval}((T_z, \tau_z), (A_z), (g_d)_{d=1}^D)$
FOR $d = 1, \ldots, D$,
  IF $g_d = 1_{G_D}$, OUTPUT $\perp$
FOR z in $V \setminus \mathcal{I}$,
  $(T_z, \tau_z) \leftarrow \mathsf{EvalGate}(z)$
FOR z in $\mathcal{O}$,
  IF $T_z \notin \{1_{G_D}, g_D\}$, OUTPUT $\perp$
OUTPUT $(\log_{g_D} T_z)_{z \in \mathcal{O}}$

proc. $\mathsf{EvalGate}(z)$
$k \leftarrow T_{L(z)} \cdot T_{R(z)}$
$d \leftarrow \mathsf{depth}(z)$
PARSE $(h_{\boldsymbol{\eta}}, u_{\boldsymbol{\eta}}, e_{\boldsymbol{\eta}}, v_{\boldsymbol{\eta}}, w_{\boldsymbol{\eta}}) \leftarrow A_z$
$\boldsymbol{\eta} \leftarrow (\tau_{L(z)}, \tau_{R(z)})$
IF $h_{\boldsymbol{\eta}} \neq g_d^k$, OUTPUT $\perp$
$(T_z, B_z) \leftarrow (e_{\boldsymbol{\eta}} / u_{\boldsymbol{\eta}}^k, w_{\boldsymbol{\eta}} / v_{\boldsymbol{\eta}}^k)$
IF $B_z \notin \{1_{G_d}, g_d\}$, OUTPUT $\perp$
OUTPUT $(T_z, \log_{g_d} B_z)$

Fig. 5. Our garbled circuit scheme with input circuit $\mathcal{C}$ of depth D and a publicly known layout. (We assume for simplicity that all edges in the circuit are from vertices of depth d to vertices of depth $d+1$ and that all output vertices have maximal depth.)

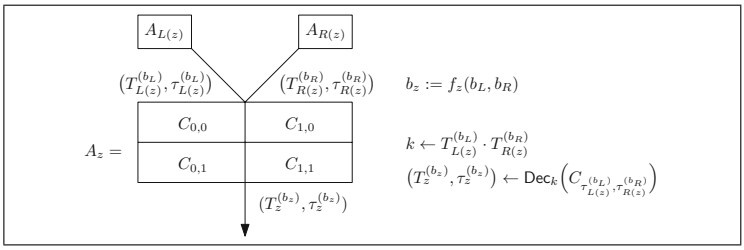

Fig. 6. A schematic representation of the evaluation of a single gate. The bits b_L, b_R, and b_z are not known to the evaluation algorithm.

Proposition 3. *The private function evaluation protocol shown in Figure 7 is correct and secure for both Alice and Bob if DDH is hard in the* (G_i).

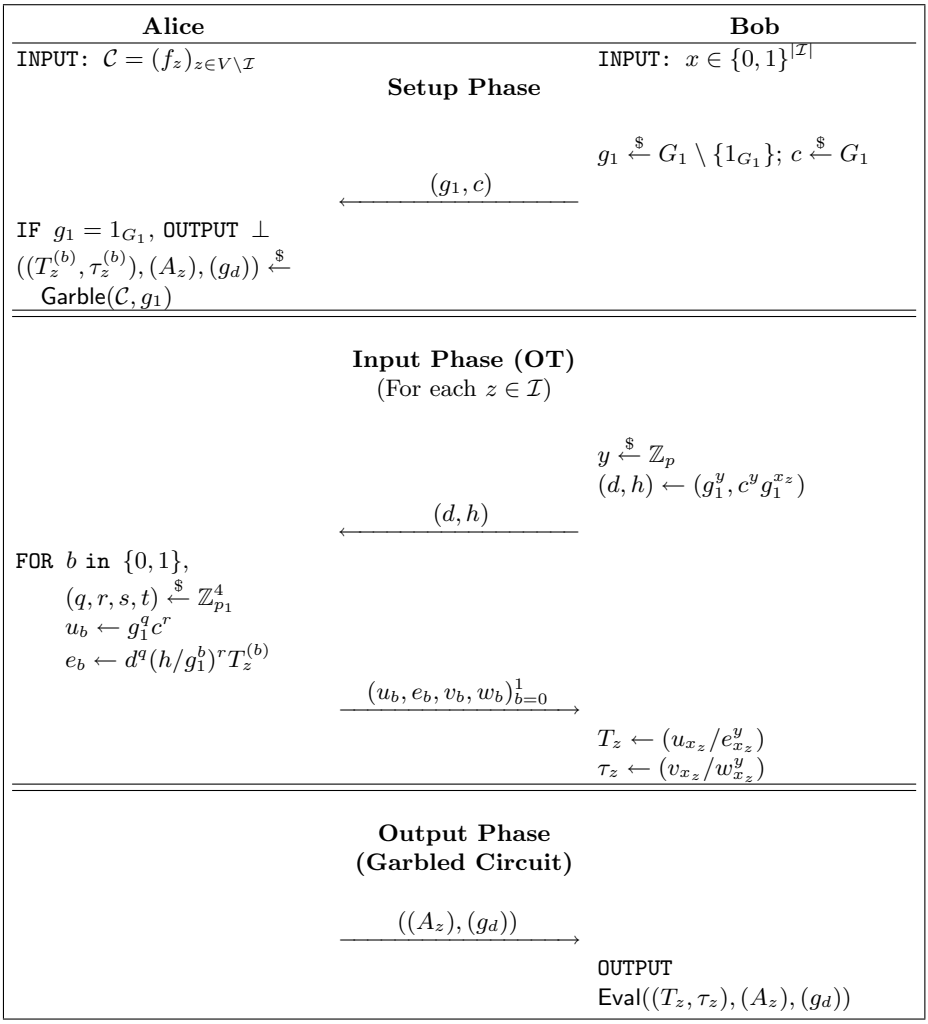

Fig. 7. A private function evaluation protocol using our oblivious transfer protocol from Section 3 and our garbled circuit scheme shown in Figure 5. (See Figure 5 for the functions Garble and Eval.)

4.3 Reverse Firewalls for PFE

Bob's reverse firewall is very similar to his reverse firewall for the oblivious transfer protocol in Section 3 (see full version for details). Alice's reverse firewall is shown in Figure 8. It makes use of a function $\mathsf{Rerand}_{\mathsf{Garble}}$ that rerandomizes garbled circuits. This procedure is rather complicated because our garbled circuits necessarily have many moving parts: location bits τ; the ordering of the ciphertexts, which is determined by the location bits; tags T; the keys, which are

products of the tags; and the randomness used to encrypt the tags and location bits. Our task is to rerandomize all of this without breaking functionality. Below, we describe the intuition behind the rerandomization procedure.

Ideally, in order to rerandomize the tags in the garbled circuit, we would simply use the malleability of ElGamal to multiply each tag $T_z^{(b)}$ by a uniformly random mask $\mathcal{R}_z^{(b)}$. But (as Gentry et al. observe in a similar context [19]), the firewall cannot know which tags are used to generate which keys—so maintaining consistency between tags and keys would not be possible with this approach. We can, however, multiply *both* $T_z^{(0)}$ and $T_z^{(1)}$ by a single uniformly random mask $\mathcal{R}_z$ (i.e., we can multiply all the corresponding ciphertexts by $\mathcal{R}_z$). Since we apply this operation to both tags, we can easily maintain consistency (after noting that an ElGamal encryption under a private key k can be easily converted into an encryption under $k \cdot k'$ without knowing k). But, we need a second degree of freedom per gate (otherwise, $T_z^{(0)}/T_z^{(1)}$ would remain unchanged).

We find our solution in the location bits. In particular, for each ciphertext $(h, u, e, v, w) = (h,\ g^r,\ h^r T,\ g^s,\ h^s g^\tau)$, we use the homomorphic property of ElGamal encryption to multiply the tag T by $g^{\beta_z \tau}$ for uniformly random β_z. Since the location bit τ encodes which ciphertexts will be encrypted under a key generated from T, we do not need to know which tags correspond to which keys to maintain consistency between the tags and keys—we just need to know that whichever tag had a corresponding location bit $\tau = 1$ was multiplied by g^{β_z}. The complete mask is therefore $\mathcal{R}_z g^{\beta_z \tau}$ for the appropriate value of τ. (Of course, a tampered implementation playing the role of Alice may not produce ciphertexts of the correct form $(h, g^r, h^r T, g^s, h^s g^\tau)$ where τ is a bit. Our rerandomization algorithm will still multiply each key of the children of the node z by the mask $\mathcal{R}_z g^{\beta_z \tau}$ for the appropriate value of τ. This rerandomization of *keys* is what makes the scheme secure.)

We also need a way to rerandomize the location bits themselves. Recall that the location bits τ are encrypted as $(v, w) = (g^s, h^s g^\tau)$. In order to rerandomize them, we note that $(v^{-1}, w^{-1} g) = (g^{-s}, h^{-s} g^{1-\tau})$ is an encryption of $\tau \oplus 1$. We can therefore flip the location bits without knowing their values.

To maintain consistency with the rerandomization of the oblivious transfer rounds, $\mathsf{Rerand}_{\mathsf{Garble}}$ takes as input the masks that should be used to rerandomize the input tags. In particular, the procedure takes as input a collection of garbled gates $(A_z)_{z \in V \setminus \mathcal{I}}$, group elements $(g_d)_{d=1}^D$, and masks for the input vertices $(\mathcal{R}_z, \beta_z, b_z^*)_{z \in \mathcal{I}}$, and it outputs new ciphertexts (A'_z) and new group elements (g'_d). The masks $\mathcal{R}_z$ and β_z are used to mask tags as described above, and the bit b_z^* determines whether the location bits $\tau_z^{(b)}$ should be flipped. (The masks for non-input vertices are selected uniformly at random by the rerandomization procedure.)

In the full version of the paper, we prove the following theorem.

Theorem 4. *The reverse firewall for Bob is robust if the DDH with a hint game is hard in* G_1.

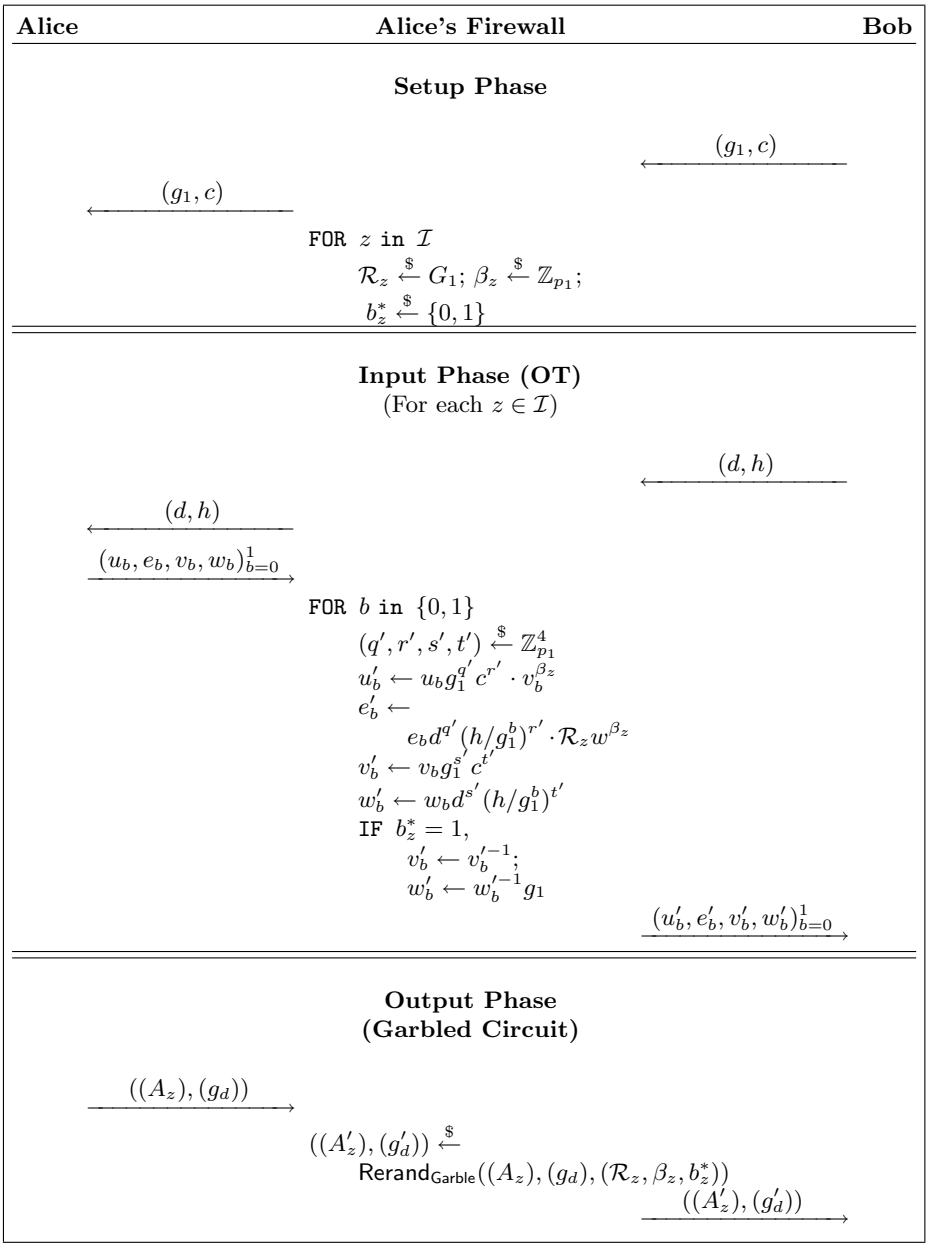

Fig. 8. Alice's firewall for the private function evaluation protocol shown in Figure 7. See the full version of the paper for the formal definition of $\mathsf{Rerand}_{\mathsf{Garble}}$.

The reverse firewall for Alice shown in Figure 8 maintains correctness, weakly preserves Alice's security, and is strongly exfiltration-resistant against an eavesdropper if non-uniform DDH is hard in the (G_d).

5 A Generic Construction for Strong Exfiltration Resistance Against Eavesdroppers

We now show that *any* protocol can be converted into a protocol that has a reverse firewall for each party that is strongly exfiltration-resistant against eavesdroppers. The resulting protocol will have at most one additional (broadcast) message per party (or fewer than two additional messages per party in the non-broadcast model). For all of the primitives that we consider in this paper, the resulting protocol will also satisfy the same security requirements as the original protocol. We cannot say that the resulting protocol will always satisfy the same security requirements for arbitrary primitives because security requirements are quite a general notion. For example, a security requirement could specifically ask that a protocol does *not* have an exfiltration-resistant reverse firewall.

In order to achieve this, the key idea is to use a public-key encryption scheme that is rerandomizable and has a *rerandomizable key*. I.e., a reverse firewall should be able to convert any public key into a uniformly random public key in such a way that it can also convert messages encrypted under the resulting key into messages encrypted under the original key. ElGamal encryption has this property (as we observe in Section 4), so we describe the scheme using ElGamal.

In particular, we interpret all messages as elements in some group G of order p in which DDH is hard. Each party computes $g \xleftarrow{\$} G \setminus \{1_G\}$ and $x \xleftarrow{\$} \mathbb{Z}_p$ and sends the message $(g, h = g^x)$ to all other parties. All future messages m sent to a party are then replaced by ciphertexts encrypted under her public key $(u = g^r, e = h^r m)$. Each time any party receives an encrypted message (u, e), she decrypts it $m = e/u^x$ and then proceeds with the protocol as normal. In addition, to prevent leakage due to early termination of the protocol, the parties never output $\perp$ until the end of the protocol; they instead send encryptions of a special message $m_\perp$ and wait until the end of the protocol to output $\perp$. A party's reverse firewall simply rerandomizes her keys and ciphertexts. If the party ever sends a message that is not of the right form, the firewall simply sends two uniformly random group elements in place of an encryption.

We show such a firewall for Alice in the two-party case in Figure 9. Note that Bob can implement essentially the same firewall. The fact that this firewall is strongly exfiltration-resistant against eavesdroppers follows immediately from the assumption that DDH is hard in G.

We note that, in general, the construction in Figure 9 should not be expected to "compose well" with other reverse firewalls. I.e., if some protocol has a reverse firewall that preserves Alice's security but is not exfiltration-resistant, we cannot necessarily apply the above transformation and obtain a protocol with a reverse firewall that *both* preserves Alice's security and is exfiltration-resistant, as it will not be possible for an efficient firewall to compute arbitrary functions on

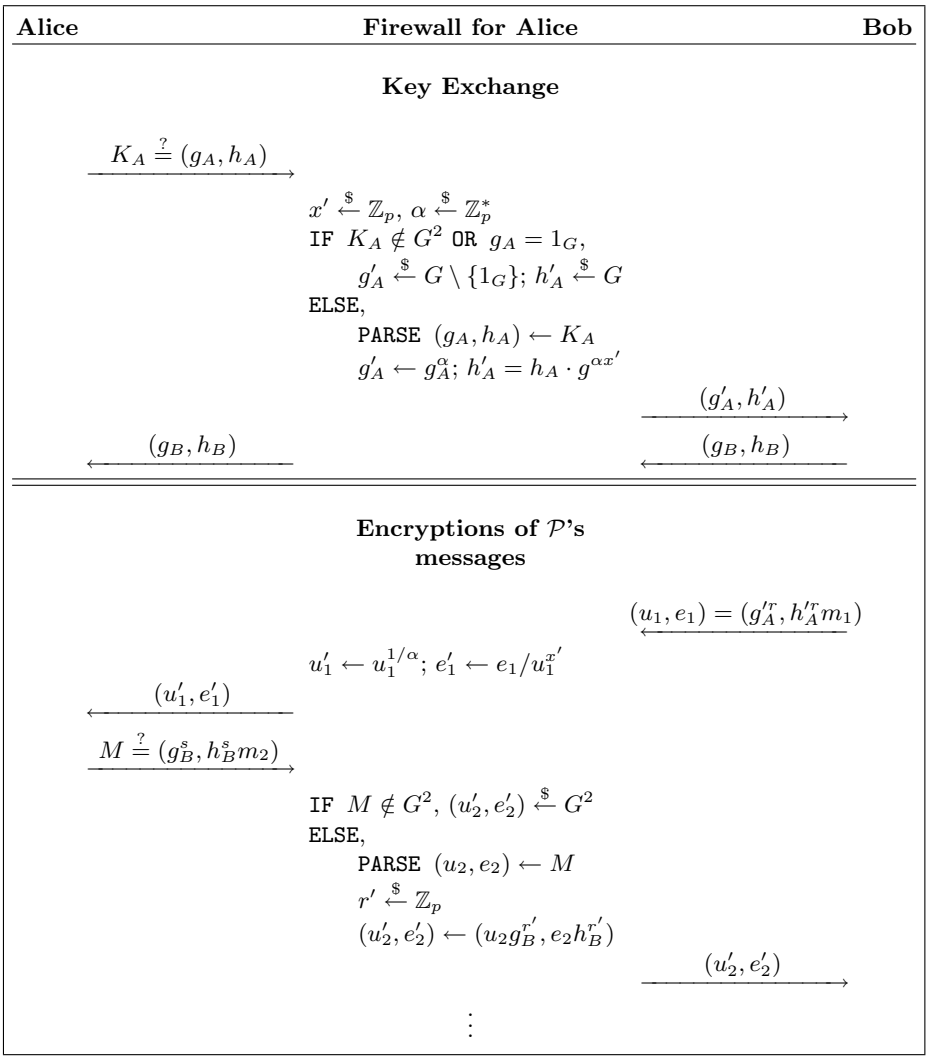

Fig. 9. A reverse firewall for Alice in a modified arbitrary two-party protocol $\mathcal{P}$. Two messages are added to the protocol in which the parties exchange public keys. They then follow the specification of $\mathcal{P}$, replacing messages m_i with ciphertexts $(g^r, h^r m_i)$. Bob has a similar reverse firewall.

the messages if they are encrypted. Even a very simple operation like equality testing (e.g., testing whether a message is some specific element) cannot be done efficiently if the message is encrypted under a semantically secure scheme. So, in general, one may need to choose between strongly exfiltration-resistant firewalls and firewalls that preserve security.

6 Conclusion and Directions for Future Work

The revelations of Edward Snowden [3,20,27] highlight a different kind of threat posed by sophisticated adversaries—the potential hijacking of a user's own software or hardware for subversive purposes. A compromised machine engaged in a cryptographic protocol may (perhaps selectively) fail to protect security or enable a covert communication channel through which the attacker can leak sensitive information or coordinate its activities. Standard solutions such as testing, auditing, or monitoring cannot in general ensure security since the attacker may use cryptographic methods to cover its tracks (aided by the complexity of modern protocols and the ubiquitous use of randomness in communications).

To counter the threat of insider attacks, we propose the concept of a (cryptographic) reverse firewall, whose role is to backstop the security of some underlying cryptographic scheme. We discuss several desirable properties of reverse firewalls (maintaining functionality, preserving security, and protecting against exfiltration attacks) and two types of tampering (arbitrary tampering and functionality-maintaining tampering). The generality of our definition provides a framework for studying insider attacks and counter-measures across a wide range of primitives.

Our main technical contribution is a protocol for private function evaluation based on Yao's garbled circuits and oblivious transfer that admits a reverse firewall for both parties. The instantiation of this remarkably strong primitive in a way that remains secure even when the user's computer has been compromised shows the power of reverse firewalls as a tool for protecting against insider attacks. In addition, our rerandomizable garbling scheme is more efficient and is secure against a stronger adversary than the scheme proposed by Gentry et al. [19] (though we rely on slightly stronger number-theoretic assumptions).

We also show that *any* protocol can be easily converted into a protocol with an exfiltration-resistant reverse firewall for each party (and the same functionality). This provides a generic way to prevent a tampered machine from leaking information to an eavesdropper via any protocol.

We conclude with a (non-exhaustive!) list of exciting directions for future work in the newly emphasized study of defense against insider attacks:

1. The most obvious direction for future work is simply the instantiation of more primitives in this framework. While this work includes an instantiation of private function evaluation (which can be used to instantiate many more primitives), there is still much more to study. For example, can we achieve stronger notions of security for two party computation? (We prove a relatively restricted notion of security for private function evaluation.) How efficiently (and under what assumptions) can we instantiate simpler primitives in this model? What can we achieve in the multi-party case? What about other primitives that are not implied by PFE (such as authenticated key agreement)?
2. We hope that future work on reverse firewalls develops a comprehensive collection of composable, efficient, modular protocols with secure reverse

firewalls. The "holy grail" would be a full characterization of functionalities and security properties for which reverse firewalls exist.
3. More generally, we hope to see a systematic study of defensive mechanisms against deliberate insider attacks. The legitimate targets of these attacks include software libraries, hardware platforms, communication channels, standards, protocols, sources of entropy, system parameters, and the choice of constants.

A Groups and Hardness Assumptions

Definition 8 (Family of groups). *We say that* $\mathbb{G} = (G_i)_{i=1}^{\infty}$ *is an* efficiently computable family of groups *if there is some probabilistic polynomial-time algorithm* setup *such that* $\mathsf{setup}(1^\lambda)$ *outputs a representation of a group* G_i *with all group elements represented by* $\mathrm{poly}(\lambda)$ *bits, a polynomial-size circuit that outputs a uniformly random group element on random input, the order of the group, and a polynomial-size circuit that computes the group operation over* G_i*.*

Throughout this paper, whenever we refer to a group G with certain properties, we implicitly define a family of groups $\mathbb{G}$ with these properties and assume that $G \xleftarrow{\$} \mathsf{setup}(1^\lambda)$, where λ is the security parameter. We assume that all algorithms have access to the group description. We write 1_G to denote the identity element in G. When we speak of negligible probabilities, polynomial-time algorithms, etc., we mean probabilities that are negligible in the security parameter λ, algorithms whose running time is polynomial in the security parameter, etc. We sometimes need to work with more than one group at a time, so we extend these notions in the natural way to a collection of groups.

Definition 9 (Decisional Diffie-Hellman). *Let* G *be a group of order* p*. Then, we say that* decisional Diffie-Hellman (DDH) is hard in G *if no probabilistic polynomial-time algorithm* $\mathcal{A}$ *can distinguish between* (g, g^x, g^y, g^{xy})*, where* $g \xleftarrow{\$} G$*,* $(x, y) \xleftarrow{\$} \mathbb{Z}_p^2$*, and* $(g_1, g_2, g_3, g_4) \xleftarrow{\$} G^4$*.*

We will need a slight variant of the DDH assumption, which we call *DDH with a hint*.

Definition 10 (DDH with a hint). *We say that* DDH with a hint is hard in G *if no probabilistic polynomial-time adversary* $\mathcal{A}$ *has non-negligible advantage in the following game.*

1. $(\sigma, g, c, d) \xleftarrow{\$} \mathcal{A}(1^\lambda)$*, with* $(g, c, d) \in G^3$*.*
2. *Sample* $b \xleftarrow{\$} \{0, 1\}$ *and* $(x, y) \xleftarrow{\$} \mathbb{Z}_p^2$*.*
3. *If* $b = 1$*, set* $z \leftarrow xy$*. Otherwise, set* $z \xleftarrow{\$} \mathbb{Z}_p$*.*
4. $b^* \xleftarrow{\$} \mathcal{A}(\sigma, (g^x, g^y, g^z, c^x, d^y))$*.*
5. $\mathcal{A}$ *wins if and only if* $b = b^*$*.*

It will also be convenient to define two hardness assumptions that are implied by DDH.

Definition 11 (Subgroup DDH). *Let G be a group of order p and $\hat{G}$ be a subgroup of $\mathbb{Z}_p^*$. We say that* $\hat{G}$-subgroup DDH *is hard in G if no probabilistic polynomial-time algorithm $\mathcal{A}$ can distinguish between (g, g^x, g^y, g^{xy}), where $g \xleftarrow{\$} G$, $(x, y) \xleftarrow{\$} \hat{G}^2$, and $(g_1, g_2, g_3, g_4) \xleftarrow{\$} G^4$.*

Lemma 1. *Let G be a group of order p and $\hat{G}$ be a subgroup of $\mathbb{Z}_p^*$. If DDH is hard in G and $|G|/|\hat{G}|$ is polynomially bounded, then $\hat{G}$-subgroup DDH is hard in G.*

Proof. Fix some probabilistic polynomial-time adversary $\mathcal{A}$ in the DDH game, and let $g \xleftarrow{\$} G$. Consider the cosets of $\hat{G}$ in $\mathbb{Z}_p^*$. In particular, we can associate to each pair of cosets (C_i, C_j) an advantage

$$P_{C_i,C_j} := \Pr_{(x,y)\xleftarrow{\$} C_i \times C_j}[\mathcal{A}(g, g^x, g^y, g^{xy}) = 1] - \Pr_{(x,y,z)\xleftarrow{\$} C_i \times C_j \times \mathbb{Z}_p}[\mathcal{A}(g, g^x, g^y, g^z) = 1] .$$

Suppose that $\mathcal{A}$ has non-negligible advantage in the $\hat{G}$-subgroup DDH game. Then, there is some constant k such that $P_{\hat{G},\hat{G}} > \lambda^{-k}$. Let $I = |G|/|\hat{G}|$ be the index of $\hat{G}$ over G, and recall that I is polynomially bounded. By the pigeonhole principle, there must be some interval between 0 and λ^{-k} of length $1/(2\lambda^k I^2)$ such that none of the values P_{C_i,C_j} is in this interval. So, by the Chernoff bound, for each (C_i, C_j), we can run $\mathcal{A}$, say, $100\lambda^{3k} I^5$ times to classify P_{C_i,C_j} as either greater than the midpoint of this interval or less than it, failing with only negligible probability. Indeed, given (g, g^x) for unknown x in coset $C(x)$ and any coset C_j, we can classify $P_{C(x),C_j}$ by running $\mathcal{A}$ a total of $100\lambda^{3k} I^5$ times on input of the form $(g, g^{\alpha x}, g^y, g^{\alpha xy})$ and input of the form $(g, g^{\alpha x}, g^y, g^z)$ for $z \xleftarrow{\$} \mathbb{Z}_p$, $\alpha \xleftarrow{\$} \hat{G}$, and $y \xleftarrow{\$} C_j$ and comparing the results. Similarly, given (g, g^y), we can classify $P_{C_i,C(y)}$.

Finally, we claim that given (g, g^x, g^y), we can classify $P_{C(x),C(y)}$. We do this by first classifying all of the P_{C_i,C_j}. We then divide the C_i into left equivalence classes such that for two elements C_i and C_k in the same equivalence class, P_{C_i,C_j} has the same classification as P_{C_k,C_j} for all C_j. We similarly divide the C_j into right equivalence classes. Finally, using the idea outlined above, we can identify the left equivalence class of $C(x)$ and the right equivalence class of $C(y)$. We can then categorize $P_{C(x),C(y)}$ by finding the unique category that "matches" these equivalence classes.

So, an adversary $\mathcal{A}'$ in the DDH game can, on input (g, g^x, g^y, g^z), first categorize $P_{C(x),C(y)}$. If $P_{C(x),C(y)}$ is greater than the midpoint of the interval, it outputs $\mathcal{A}(g, g^x, g^y, g^z)$. Otherwise, it flips a coin and outputs the result. Since $P_{\hat{G},\hat{G}} > \lambda^{-k}$, it follows that with probability at least $|\hat{G}|/|G| = 1/\text{poly}(\lambda)$, we have that $P_{C(x),C(y)}$ is larger than the midpoint. The result follows. □

Definition 12 (k-DDH and subgroup k-DDH). *Let G be a group of order p. For $k \geq 2$, we say that* k-DDH is hard in G *if no probabilistic polynomial-time algorithm $\mathcal{A}$ can distinguish between $(g, (g^{x_i})_{i=1}^k, (g^{x_i x_j})_{1 \leq i < j \leq k})$ and $(g_i^*)_{i=1}^\ell$ with $\ell = k(k+1)/2 + 1$, where $g \xleftarrow{\$} G$, $(x_i) \xleftarrow{\$} \mathbb{Z}_p^k$, and $(g_i^*) \xleftarrow{\$} G^\ell$.*

Let G be a group of order p, $\hat{G}$ be a subgroup of $\mathbb{Z}_p^*$, and $k \geq 2$. We say that $\hat{G}$-subgroup k-DDH is hard in G *if no probabilistic polynomial-time algorithm $\mathcal{A}$ can distinguish between $(g, (g^{x_i})_{i=1}^k, (g^{x_i x_j})_{1 \leq i < j \leq k})$ and $(g_i^*)_{i=1}^\ell$, where $g \xleftarrow{\$} G$, $(x_i) \xleftarrow{\$} \hat{G}^k$, and $(g_i^*) \xleftarrow{\$} G^\ell$.*

Lemma 2. *Let G be a group of order p. If DDH is hard in G, then k-DDH is hard in G for any polynomially bounded k.*

Let $\hat{G}$ be a subgroup of $\mathbb{Z}_p^*$. If DDH is hard in G and $|G|/|\hat{G}|$ is polynomially bounded, then $\hat{G}$-subgroup k-DDH is hard in G for any polynomially bounded k.

Proof. For $i = 0, \ldots, (k^2 - k)/2$, let **Game** i be the game of distinguishing between a uniformly random tuple $(g_1^*, \ldots, g_\ell^*) \xleftarrow{\$} G^\ell$ and a tuple of the form $(g, (g^{x_i})_{i=1}^k, (g^{x_i x_j})_{1 \leq i < j \leq k})$ with the last i elements changed to a uniformly random element. It follows from the assumption that DDH is hard in G that no adversary can have non-negligibly larger advantage in **Game** i than in **Game** $i+1$. The result follows by noting that **Game 0** is the k-DDH game and that no adversary can have any advantage in **Game** $(k^2 - k)/2$. The second claim follows analogously to the proof of Lemma 1. □

We will also require a non-uniform version of DDH.

Definition 13 (Non-uniform decisional Diffie-Hellman). *Let G be a group of order p. We say that* non-uniform decisional Diffie-Hellman is hard in G *if no probabilistic polynomial-time algorithm $\mathcal{A}$ with* auxiliary information $\mathsf{aux} = \mathsf{aux}(G)$ *can distinguish between (g, g^x, g^y, g^{xy}), where $g \xleftarrow{\$} G$, $(x, y) \xleftarrow{\$} \mathbb{Z}_p^2$, and $(g_1, g_2, g_3, g_4) \xleftarrow{\$} G^4$. Note that* aux *does not need to be efficiently computable.*

We similarly extend the definitions of $\hat{G}$-subgroup DDH, k-DDH, and $\hat{G}$-subgroup k-DDH to the non-uniform setting. Finally, one more definition will be useful.

Definition 14 (Chosen-bases $\hat{G}$-subgroup k-DDH). *Let G be a group of order p, $\hat{G}$ be a subgroup of $\mathbb{Z}_p^*$, and $k \geq 2$. We say that* chosen-bases $\hat{G}$-subgroup k-DDH is hard in G *if no probabilistic polynomial-time algorithm $\mathcal{A}$ has non-negligible advantage in the following game.*

1. $(\sigma, (g_i)_{i=1}^k, (h_{i,j})_{1 \leq i < j \leq k}) \xleftarrow{\$} \mathcal{A}(1^\lambda)$, *with* $g_i, h_{i,j} \in G \setminus \{1_G\}$.
2. *Sample* $b \xleftarrow{\$} \{0,1\}$, $(x_i)_{i=1}^k \xleftarrow{\$} \mathbb{Z}_p^k$, *and* $(g_i^*)_{i=1}^\ell \xleftarrow{\$} G^\ell$, *where* $\ell = (k^2 + k)/2$.
3. *If* $b = 0$, $b^* \xleftarrow{\$} \mathcal{A}(\sigma, (g_i^{x_i})_{i=1}^k, (h_{i,j}^{x_i x_j})_{i<j})$. *Otherwise,* $b^* \xleftarrow{\$} \mathcal{A}(\sigma, (g_i^*))$.
4. $\mathcal{A}$ *wins if and only if* $b = b^*$.

Lemma 3. *Let G be a group of order p, and let $\hat{G}$ be a subgroup of $\mathbb{Z}_p^*$. If non-uniform DDH is hard in G and $|G|/|\hat{G}|$ is polynomially bounded, then chosen-bases $\hat{G}$-subgroup k-DDH is hard in G for any polynomially bounded k.*

Proof. We first note that the natural non-uniform analogue of Lemma 2 holds by an essentially identical proof. In particular, it suffices to show that chosen-bases $\hat{G}$-subgroup k-DDH is hard in G if non-uniform $\hat{G}$-subgroup k-DDH is hard in G.

Let $\mathcal{A}$ be an adversary in the chosen-bases $\hat{G}$-subgroup k-DDH game in G. Note that $\mathcal{A}$ may not be deterministic, but we can fix the output of $\mathcal{A}$, $(\sigma, (g_i)_{i=1}^k, (h_{i,j})_{1\le i<j\le k}) \xleftarrow{\$} \mathcal{A}(1^\lambda)$, such that the advantage of $\mathcal{A}$ with this fixed output is maximal. Let $\mathsf{aux} = (\sigma, (\log_{g_1}(g_i))_{i=1}^k, (\log_{g_1}(h_{i,j}))_{1\le i<j\le k})$.

We then build $\mathcal{A}'$, an adversary in the non-uniform $\hat{G}$-subgroup k-DDH in G as follows. $\mathcal{A}'$ receives auxiliary input $(\sigma, (\log_{g_1}(g_i))_{i=1}^k, (\log_{g_1}(h_{i,j}))_{1\le i<j\le k})$ and challenge $((g_i^*)_{i=1}^k, (h_{i,j}^*))$. For each i, j, it sets $g_i' \leftarrow g_i^{*\log_{g_1} g_i}$ and $h_{i,j}' \leftarrow h_{i,j}^{*\log_{g_1}(h_{i,j})}$. It then returns $\mathcal{A}(\sigma, (g_i'), (h_{i,j}'))$.

It should be clear that the view of $\mathcal{A}$ is identical to its view in the $\hat{G}$-subgroup k-DDH game in G. □

References

1. Aiello, W., Ishai, Y., Reingold, O.: Priced oblivious transfer: how to sell digital goods. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 119–135. Springer, Heidelberg (2001)
2. Alwen, J., Shelat, A., Visconti, I.: Collusion-free protocols in the mediated model. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 497–514. Springer, Heidelberg (2008)
3. Ball, J., Borger, J., Greenwald, G.: Revealed: how US and UK spy agencies defeat internet privacy and security. Guardian Weekly, September 2013
4. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS 2012, pp. 784–796. ACM, New York (2012)
5. Bellare, M., Paterson, K.G., Rogaway, P.: Security of symmetric encryption against mass surveillance. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 1–19. Springer, Heidelberg (2014)
6. Bellare, M., Paterson, K.G., Rogaway, P.: Security of symmetric encryption against mass surveillance. Cryptology ePrint Archive, Report 2014/438 (2014). http://eprint.iacr.org/
7. Blaze, M., Bleumer, G., Strauss, M.J.: Divertible protocols and atomic proxy cryptography. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 127–144. Springer, Heidelberg (1998)
8. Brown, D., Vanstone, S.: Elliptic curve random number generation, US Patent App. 11/336,814, August 16 (2007)

9. Burmester, M., Desmedt, Y.G.: All languages in NP have divertible zero-knowledge proofs and arguments under cryptographic assumptions. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 1–10. Springer, Heidelberg (1991)
10. Burmester, M., Desmedt, Y., Itoh, T., Sakurai, K., Shizuya, H.: Divertible and subliminal-free zero-knowledge proofs for languages. J. Cryptology **12**, 197–223 (1999)
11. Burmester, M., Desmedt, Y., Itoh, T., Sakurai, K., Shizuya, H., Yung, M.: A progress report on subliminal-free channels. In: Anderson, R. (ed.) Information Hiding. LNCS, vol. 1174, pp. 157–168. Springer, Berlin Heidelberg (1996)
12. Vulnerability summary for CVE-2014-1260 ('Heartbleed'), April 2014. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1260
13. Vulnerability summary for CVE-2014-1266 ('goto fail'), February 2014. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1266
14. Vulnerability summary for CVE-2014-6271 ('Shellshock'), September 2014. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271
15. Desmedt, Y.: Subliminal-free sharing schemes. In: Proceedings of the 1994 IEEE international symposium on information theory, p. 490, June 1994
16. Desmedt, Y.G.: Abuses in cryptography and how to fight them. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 375–389. Springer, Heidelberg (1990)
17. ElGamal, Taher: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Robert Blakley, George, Chaum, David (eds.) CRYPT 2004. LNCS, vol. 196, pp. 10–18. Springer, New York (1985)
18. Ford, K., Konyagin, S.V., Luca, F.: Prime chains and Pratt trees. Geometric and Functional Analysis **20**(5), 1231–1258 (2010)
19. Gentry, C., Halevi, S., Vaikuntanathan, V.: i-Hop homomorphic encryption and rerandomizable yao circuits. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 155–172. Springer, Heidelberg (2010)
20. Greenwald, G.: No Place to Hide: Edward Snowden, the NSA, and the U.S. Surveillance State. Metropolitan Books, May 2014
21. Harnik, D., Kilian, J., Naor, M., Reingold, O., Rosen, A.: On robust combiners for oblivious transfer and other primitives. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 96–113. Springer, Heidelberg (2005)
22. Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T., Wachter, C.: Public keys. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 626–642. Springer, Heidelberg (2012)
23. Lepinksi, M., Micali, S., Shelat, A.: Collusion-free protocols. In: Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing, STOC 2005, pp. 543–552. ACM, New York (2005)
24. Mironov, I., Stephens-Davidowitz, N.: Cryptographic reverse firewalls. Cryptology ePrint Archive, Report 2014/758, full version (2014). http://eprint.iacr.org/
25. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2001, pp. 448–457. Society for Industrial and Applied Mathematics, Philadelphia (2001)
26. Okamoto, T., Ohta, K.: Divertible zero knowledge interactive proofs and commutative random self-reducibility. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 134–149. Springer, Heidelberg (1990)
27. Perlroth, N., Larson, J., Shane, S.: N.S.A. able to foil basic safeguards of privacy on Web. The New York Times, September 2013

28. Prabhakaran, M., Rosulek, M.: Rerandomizable RCCA encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 517–534. Springer, Heidelberg (2007)
29. Shumow, D., Ferguson, N.: On the possibility of a back door in the NIST SP800-90 Dual Ec Prng. CRYPTO Rump Session (2007)
30. Simmons, G.: The prisoners' problem and the subliminal channel. In: Chaum, D. (ed.) Advances in Cryptology, pp. 51–67. Springer, US (1984)
31. Simmons, G.J.: The subliminal channel and digital signatures. In: Beth, T., Cot, N., Ingemarsson, I. (eds.) EUROCRYPT 1984. LNCS, vol. 209, pp. 364–378. Springer, Heidelberg (1985)
32. Young, A., Yung, M.: The dark side of "Black-Box" cryptography, or: should we trust capstone? In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 89–103. Springer, Heidelberg (1996)