Message Transmission with Reverse Firewalls— Secure Communication on Corrupted Machines

Yevgeniy Dodis[1*], Ilya Mironov[2], and Noah Stephens-Davidowitz[1**]

[1] Dept. of Computer Science, New York University.
[2] Google.

Abstract. A secure reverse firewall, as recently defined by Mironov and Stephens-Davidowitz, is a third party that "sits between a user and the outside world" and modifies the user's sent and received messages so that *even if the user's machine has been corrupted*, her security is still guaranteed. In other words, reverse firewalls allow us to provide meaningful (and, indeed, very strong) security guarantees against powerful adversaries that may have tampered with the user's hardware or software (or adversaries that are aware of bugs in the user's protocol implementation). A long list of recent revelations shows that such threats are extremely common in practice, and they present a serious, arguably existential, threat to cryptography. Importantly, reverse firewalls defend against such threats without sharing any secrets with the user, and in general we expect the user to place essentially no trust in the firewall.

While Mironov and Stephens-Davidowitz demonstrated that reverse firewalls can be constructed for very strong cryptographic primitives (which are of mostly theoretical interest), we study reverse firewalls for perhaps the most natural cryptographic task: secure message transmission. We find a rich structure of solutions that vary in efficiency, security, and setup assumptions, in close analogy with message transmission in the classical setting. Our strongest and most important result shows a protocol that achieves interactive, concurrent CCA-secure message transmission with a reverse firewall—i.e., CCA-secure message transmission on a possibly compromised machine! Surprisingly, this protocol is quite efficient and simple, requiring only a small constant number of public-key operations. It could easily be used in practice. Behind this result is a technical composition theorem that shows how key agreement with a sufficiently secure reverse firewall can be used to construct a message-transmission protocol with its own secure reverse firewall.

1 Introduction

In the past few years, the cryptographic community has learned of a disturbingly wide array of new security vulnerabilities. The revelations of Edward Snowden show that the United States National Security Agency successfully gained access to secret information by extraordinary means, including subverting cryptographic standards [PLS13, BBG13] and intercepting and tampering with hardware on its way to users [Gre14]. Meanwhile, many (apparently accidental) security flaws have been found in widely deployed pieces of cryptographic software, leaving users completely exposed [LHA$^+$12, CVE14b, CVE14a, CVE14c]. Due to the complexity of modern cryptographic software, such vulnerabilities are extremely hard to detect in practice, and, ironically, cryptographic modules are often the easiest to attack, as attackers can often use cryptographic mechanisms to mask their activities or opportunistically hide their communications within encrypted traffic (as in the case of the Heartbleed vulnerability). This leads to cryptography's current existential crisis, summarized by the following (terrifying) question: "How can we provide any meaningful security guarantees when the adversary may have arbitrarily tampered with its victim's computer?"

Motivated by these concerns, Mironov and Stephens-Davidowitz recently proposed the novel concept of *(cryptographic) reverse firewalls*, designed to protect machines from *insider* attacks [MS15]. Reverse firewalls are autonomous (untrusted) intermediaries that intercept and "sanitize" transit traffic to backstop security of

* Partially supported by gifts from VMware Labs and Google, and NSF grants 1319051, 1314568, 1065288, 1017471.
** Partially supported by National Science Foundation under Grant No. CCF-1320188. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

network clients. A cryptographic protocol equipped with a reverse firewall has the desirable guarantee that its security is preserved even if it is run on a compromised machine. In other words, a correctly implemented reverse firewall ensures the security of a cryptographic protocol *even if the user's own implementation of the protocol is buggy, or even maliciously compromised*. Such a tool is a natural and powerful solution to the problems discussed above. We stress that the firewall is untrusted and shares no secrets with the user.

More concretely, Mironov and Stephens-Davidowitz start by considering an arbitrary cryptographic protocol that satisfies some notions of security and functionality (i.e., correctness). For example, perhaps the simplest non-trivial case is semantically secure message transmission from Alice to Bob, which has the functionality requirement that Bob should receive Alice's plaintext message and the security requirement that a computationally bounded adversary "should not learn anything about Alice's plaintext message" from the transcript of a run of the protocol. Formally, we can model this functionality by providing Alice with an arbitrary input plaintext and requiring that Bob's output match this input,[1] and we can model semantic security by a standard indistinguishability security game. A reverse firewall for Alice in such a protocol is a party that "sits between" Alice and the outside world and modifies the messages that Alice sends and receives. Such a firewall *maintains functionality* if the resulting protocol achieves the same functionality as the original. E.g., in the case of message transmission, Bob's output should still match Alice's input plaintext. More interestingly, the firewall *preserves security* if the protocol remains secure when we replace Alice's computer with some arbitrarily corrupted party "composed with" the firewall. For example, a reverse firewall for Alice preserves semantic security of message transmission if a computationally bounded adversary "learns nothing about Alice's plaintext message" from the transcript of messages sent between the firewall and Bob, *regardless of how Alice behaves*. (E.g., the firewall may rerandomize the messages that Alice sends in a way that makes them indistinguishable from random from the adversary's perspective, regardless of Alice's original message. We analyze such protocols in a stronger setting in Section 3.1.) Finally, the reverse firewall *resists exfiltration* if a compromised implementation "cannot leak any information to the outside world" beyond the functionality of the protocol.[2] (See Section 2.1 for the formal definitions and [MS14] for a thorough discussion.)

It is convenient to identify a class of *functionality-maintaining adversaries*: compromised implementations that are "technically legal" in the sense that they don't break the protocol but otherwise can deviate arbitrarily. Some of our reverse firewalls are only secure against this type of corruption. (This model is introduced by [MS15], and the authors call security against unrestricted compromise *strong* security.) We emphasize that this restricted class of compromised implementations is still quite large. In particular, all of the real-world compromises mentioned above fall into this category [PLS13, BBG13, Gre14, Sup15, LHA+12, CVE14b, CVE14a, CVE14c], as do essentially all other forms of compromise considered in prior work, such as faulty or backdoored PRNGs [DGG+15], Algorithm Substitution Attacks [BPR14a], subliminal channels [Sim84], etc. (See [MS14] for a detailed comparison of the reverse firewall framework with prior work.)

Mironov and Stephens-Davidowitz demonstrate feasibility of the framework by constructing reverse firewalls for parties participating in Oblivious Transfer and Secure Function Evaluation protocols—very strong cryptographic primitives. However, these protocols are very inefficient and mostly of theoretical interest. To fulfill the promise of reverse firewalls, we need to consider protocols of practical importance, which is the focus of the present work. We demonstrate efficient (i.e., using a small constant number of public-key operations) reverse firewalls for perhaps the most natural cryptographic task: secure message transmission. Our strongest protocols provide security guarantees in the reverse firewall setting comparable to the strongest guarantees obtained in the classical setting (e.g., security against adaptive chosen ciphertext attacks with forward secrecy). Surprisingly, our protocols are also relatively efficient, requiring only a few rounds and a small constant number of public-key operations. To achieve this, we also provide key-agreement protocols with efficient, secure reverse firewalls, which may be of independent interest. We group our solutions accord-

[1] Note that Mironov and Stephens-Davidowitz's notion of functionality is quite simple. Formally, they define a functionality requirement as any condition on the output of the parties that may depend on the input, and in practice, these requirements are straightforward. The reader should not confuse this with the more complicated definition of functionality used in the universal composability framework.

[2] Often, exfiltration resistance is implied by security preservation, but not always. For example, a reverse firewall for Alice in a message transmission protocol preserves semantic security if and only if it is exfiltration resistant. However, semantic security does not guarantee that Bob will not leak any information, but it still may be desirable for Bob to have a reverse firewall in such a protocol that resists exfiltration.

ing to a variety of setup assumptions: the parties share a secret key; there is no public-key infrastructure; both parties have a public-key/private-key pair; and only the recipient has a public-key/private-key pair. Non-trivial levels of security are achievable in almost all cases, and the various schemes mirror closely those that appear in the classical setting.

1.1 Related work

Message transmission in the classical setting (i.e., without reverse firewalls) is extremely well-studied, and a summary of such work is beyond the scope of this paper. (We note, however, that our notions of security of interactive message transmission protocols follow closely Dodis and Fiore [DF14].) Similarly, there have been many different approaches to cryptography in the presence of compromise. We therefore provide only a limited discussion of related work. (Both [BPR14a] and [MS15] contain a more thorough discussion of alternative approaches to security in the presence of a compromise.)

Bellare, Paterson, and Rogaway consider the problem of securely encrypting a message when the parties may be compromised [BPR14a]. However, they do not allow for reverse firewalls (as their work predates the RF framework). As a result, they necessarily restrict themselves to the case in which the parties share a secret key and the compromised party maintains functionality. (I.e., a tampered encryption algorithm must produce ciphertexts that decrypt to the correct plaintext.) Their main result is a lower bound, showing roughly that even a relatively weak adversary can break any scheme that "non-trivially uses randomness" in this setting. (Of course, this lower bound no longer applies in the reverse firewalls framework, as our results demonstrate.) They also provide a positive result that is more relevant to our work, in which they show how to build a CCA-secure deterministic stateful scheme. The main idea behind their positive result is the observation that, if an encryption scheme is deterministic and has a unique ciphertext that decrypts to each plaintext, then a tampered implementation of the encryption algorithm can only maintain functionality if its behavior is identical to the honest encryption algorithm. In short, they show that "randomness is dangerous" in this setting.

Our work can be viewed as a generalization of [BPR14a] in a number of directions. We consider multi-round protocols in which the parties may not share secret keys, and we consider arbitrarily compromised adversaries. In order to do this, we use the reverse firewalls framework of [MS15]. While Bellare et al.'s work stresses the danger of randomness in secure message transmission, our work highlights the benefits of randomness. In particular, our schemes rely heavily on "rerandomization" by the reverse firewall. However, we do use the deterministic encryption scheme of Bellare et al. as part of two of our protocols.

Recently, Ateniese, Magri, and Venturi studied reverse firewalls for signature schemes [AMV15]. Their work can be considered as complementary to ours, as we are concerned with privacy, while they consider authentication. We also note that our more advanced key-agreement scheme uses unique signatures, and we implicitly rely on the fact that unique signatures have a reverse firewall, as [AMV15] prove. Indeed, the more general primitive of rerandomizable signatures that they consider would also suffice for our purposes.

Finally, we note that some of our study of key agreement is similar to work on key-agreement protocols secure against active insiders, and the study of key control (e.g., [PW03, KS05, DPSW06]). These works consider key-agreement protocols involving at least three parties, in which one or more of the participants wishes to maliciously fix the key or otherwise subvert the security of the protocol. Some of the technical challenges that we encounter are similar to those encountered in the key control literature, and indeed, the simple toy protocol that we present in Section 3.3 can be viewed as a simple instantiation of some of the known solutions to the key-control problem (e.g., [DPSW06]). However, since prior work approached this problem from a different perspective—with three or more parties and without reverse firewalls—our more technical solutions presented in Section 4.2 are quite different. In particular, the key-control literature typically focuses on creating protocols that produce "non-malleable" keys, whereas our protocols are designed specifically to allow the firewall to maul the keys (without sacrificing security).

1.2 Our results

Secure transmission of a message between two parties is arguably the most basic and essential cryptographic task. In this section, we walk through several common setup assumptions for this problem and outline our solutions for secure message-transmission protocols with reverse firewalls against end-point compromises. In what follows, Alice is always the sender and Bob is always the receiver of the message. All of our security notions apply to the concurrent setting, with potentially many runs of the protocol running simultaneously.

Symmetric-key setting. In the first and simplest scenario, Alice and Bob have a shared secret key. (See Appendix A.) Quite naturally, Alice may want to use a symmetric-key encryption scheme to communicate with Bob. Using a standard scheme (e.g., CBC-AES) would, however, expose her to a number of algorithm-substitution attacks (ASA) described by Bellare, Paterson, and Rogaway [BPR14a], such as IV-replacement or biased-ciphertext attacks. To defend against ASA Bellare et al. propose using a *stateful, deterministic* encryption scheme based on either a counter or a nonce, as we described above. We briefly consider this case, observing that their solution corresponds to a one-round protocol in our model (in which the firewall simply lets messages pass unaltered). We also show that strong security (when Alice's implementation is not necessarily functionality-maintaining) is not achievable within this framework without using (less efficient) public-key primitives.

Unkeyed setting. In the next scenario, Alice and Bob have no shared secret keys and no pre-existing public-key infrastructure. (See Section 3.) This is the first setting in which reverse firewalls meaningfully improve security. Since neither the sender nor the receiver can be authenticated in this setting, the strongest guarantee achievable in this scenario is security against passive adversaries, i.e. security against chosen-plaintext attacks (CPA). While this level of security is typically considered to be insufficient in practice, the ideas that we develop here will be useful for solving the harder problems that we discuss below. As such, we present two solutions in this setting.

First, we consider a simple two-round protocol in which Bob sends a (freshly generated) public key and Alice responds with an encryption of her message under Bob's key. Of course, if Bob's computer is compromised, then the key that he sends can be used as a channel to leak his secrets—either to Alice or an eavesdropper. Bob's reverse firewall "plugs" this channel by rerandomizing the public key and undoing the transformation when it receives Alice's response. Similarly, if Alice's computer is compromised, then her ciphertext can be a vulnerability—it may be used as a channel to leak a secret, or it may be encrypted using "bad randomness" or otherwise improperly, allowing an eavesdropper to read her plaintext message. Alice's reverse firewall plugs this hole by rerandomizing her ciphertext. (We note that both the keys and ciphertexts used in ElGamal encryption can easily be rerandomized in the way that we require here.) Indeed, both reverse firewalls guarantee *strong* security for Alice and Bob—even if the players' implementations produce malformed messages or just refuse to cooperate, their respective firewalls can still guarantee security.

The above solution is simple and elegant, but such protocols are of little practical value because they require the computation of public-key operations on the entire plaintext. Since plaintexts are often quite long and public-key operations tend to be much slower than symmetric-key operations, this can be quite inefficient. In practice, it is much faster to use public-key operations to transmit a (relatively short) key for a symmetric-key encryption scheme and then send the message encrypted under this symmetric key. There are two general methods for transmitting this key in the classical setting: hybrid encryption and key agreement.

Before we describe our solution based on key agreement, it will be instructive to consider why hybrid encryption cannot work in this setting. Recall that in a hybrid encryption scheme, Bob sends a public key to Alice (as above), Alice selects a uniformly random key rk for a symmetric-key scheme and sends rk encrypted under Bob's public key together with the encryption of her message under rk. Such a scheme fails in our setting because a corrupted implementation of Alice can "choose a bad key" rk^* with which to encrypt the message. The "bad key" rk^* may be known to an adversary; may be chosen so that the ciphertext takes a specific form that leaks some information; or may itself be used to leak additional information to Bob. Intuitively, a firewall cannot hope to fix this problem (without resorting to slow public-key primitives).

Luckily, key agreement can be combined with symmetric-key encryption to securely and efficiently transmit a message in the setting of reverse firewalls. A key-agreement protocol allows Alice and Bob to exchange a secret key over an insecure channel. Such a protocol is often used in conjunction with symmetric-key encryption in the classical setting, where it is justified by composition theorems relating the security of the message-transmission protocol to the underlying key-agreement protocol. Indeed, we give an analogous result (Theorem 2) that works in our setting, showing that a key-agreement protocol with sufficiently secure reverse firewalls can be combined with symmetric-key encryption to produce an efficient CPA-secure message-transmission protocol with secure reverse firewalls. This motivates the study of key agreement in the setting of reverse firewalls, and in Section 3.3, we show how to construct a relatively simple key-agreement protocol with reverse firewalls that suffices. (Intuitively, the reason that this solution does not suffer the same fate as hybrid encryption in this setting is because it uses interaction, which crucially happens through the reverse firewall, to "prevent either party from controlling the value of the key.")

Publicly keyed setting. In the stronger setting in which we have a public-key infrastructure, with a public-key/private-key pair for each party (considered in Section 4), we can hope to achieve interactive CCA security (i.e., the adversary may interact with the parties arbitrarily and may view Bob's output). Perhaps surprisingly, we prove a generic composition theorem in this setting, which shows that it suffices to construct a key-agreement protocol with reverse firewalls that satisfy certain security properties. In analogy with the unkeyed setting, after agreeing to a key with Bob, Alice uses symmetric-key encryption to send the actual plaintext message. The resulting protocol is CCA-secure, and Alice's reverse firewall preserves this security. (See Theorem 4.)

To instantiate this scheme, we then construct (in Section 4.2) a key-agreement protocol that satisfies the necessary security properties. In particular, the scheme is secure against active adversaries, and each party has a reverse firewall that preserves this security. This key-agreement protocol is significantly more technical than our scheme in the unkeyed setting, and common techniques from classical key agreement do not work. In particular, many key-agreement protocols achieve security against an active adversary by having both parties sign the transcript at the end of the protocol; intuitively, this allows the parties to know if the adversary has tampered with any messages, so that they will never agree to different keys or a key chosen dishonestly. But in our setting, we actually *want* the firewall to be able to modify the parties' messages! We therefore need to find some unique information that the parties can use to confirm that they have agreed to the same key without preventing the firewall from modifying with the parties'. Furthermore, we need the firewall to be able to check signatures itself, so that it can block invalid messages. Therefore, our primary technical challenge in this context is to find some piece of information that (1) uniquely identifies the key; (2) respects the firewall's changes to the parties' messages; and (3) is efficiently computable from the transcript. And, of course, the key itself must still be indistinguishable from random in the presence of this information. Our solution uses hashed Diffie-Hellman (similar in spirit to [Kil07]) and bilinear maps. (We also use unique signatures to prevent the signatures from becoming a channel themselves.)

Surprisingly, in spite of the challenges, our protocol only requires four rounds with relatively short messages, and the parties themselves (including the firewall) only need to perform a small constant number of operations.[3] This compares quite favorably with protocols that are currently implemented in practice (which of course are not secure in our setting), and we therefore believe that this protocol can and should be implemented and used in the real world.

We highlight an addition technical point. Observe that Bob's (the receiver's) reverse firewall is only able to modify Bob's *messages*, but not his output. (Indeed, since we place no trust in the firewall, it should not have access to Bob's private output.) In the CCA security game in our setting, in which an active adversary is allowed to request output from a possibly corrupt implementation of Bob, this is a severe handicap. Syntactically and conceptually, the reverse firewall processes Bob's messages that reach "the wire" but not its outputs, which are internal to Bob's software stack and are therefore not available to the firewall. So, consider a compromised implementation that, in response to some unrelated run of the protocol, spits out the challenge message to its caller. The firewall may never see (nor should it see) how the output of the message-transmission protocol is consumed. The adversary, on the other hand, may observe Bob's behavior and win the CCA game. As such, we are only interested in firewalls that maintain CCA security for Alice or CPA-security for Bob. We note that this issue seems to be inherent.

Singly keyed setting. In the "intermediate" setting in which Bob has a public-key/private-key pair but Alice does not, we show how to achieve CCA security in a single round using rerandomizable RCCA-secure encryption [Gro04, PR07]. (See Section 5.) However, these schemes are not forward secret, and they require relatively inefficient public-key primitives that must be applied to full plaintexts. We leave to future work the question of whether CCA-secure and forward-secret message transmission is achievable in this regime.

[3] For example, Bob only needs to perform one exponentiation over a group in which DDH is hard, compute a bilinear map once, check the opening of a commitment, verify one (unique) signature, (uniquely) sign one string, and apply a hash function once.

We also do not know whether a composition theorem similar to Theorem 4 exists in this regime. (We discuss some additional open questions in Section 6.)

2 Definitions

2.1 Reverse firewalls

We use the definition of a reverse firewalls from [MS14] (and we refer the reader to [MS14] for a longer discussion of the reverse firewall framework). In particular, a reverse firewall $\mathcal{W}$ is just a stateful algorithm that maps messages to messages. For a party A and reverse firewall $\mathcal{W}$, we define $\mathcal{W} \circ \mathsf{A}$ as the "composed" party in which $\mathcal{W}$ is applied to the messages that A receives before A "sees them" and the messages that A sends before they "leave the local network of A." $\mathcal{W}$ has access all public parameters, but not to the private input of A or the output of A. We repeat all relevant definitions from [MS14] below, and we add two new ones.

As in [MS14], we assume that protocols come with some functionality or correctness requirements $\mathcal{F}$ and security requirements $\mathcal{S}$. (For example, a functionality requirement $\mathcal{F}$ might require that Alice and Bob output the same thing at the end of the protocol. A security requirement might ask that no efficient adversary can distinguish between the transcript of the protocol and uniformly random strings.) Throughout, we use $\bar{\mathsf{A}}$ to represent arbitrary adversarial implementations of party A and $\widetilde{\mathsf{A}}$ to represent functionality-maintaining adversarial implementations of A (i.e., implementations of A that still satisfy the functionality requirements of the protocol.) For a protocol $\mathcal{P}$ with party A, we write $\mathcal{P}_{\mathsf{A}\to\widetilde{\mathsf{A}}}$ to represent the protocol in which the role of party A is replaced by party $\widetilde{\mathsf{A}}$.

We are only interested in firewalls that themselves maintain functionality. In other words, the *composed party* $\mathcal{W} \circ \mathsf{A}$ should not break the correctness of the protocol. (Equivalently, $\mathcal{P}_{\mathsf{A}\to\mathcal{W}\circ\mathsf{A}}$ should satisfy the same functionality requirements as the underlying protocol $\mathcal{P}$.) We follow [MS15] in requiring something slightly stronger—reverse firewalls should be "stackable", so that many reverse firewalls composed in series $\mathcal{W} \circ \cdots \circ \mathcal{W} \circ \mathsf{A}$ still do not break correctness. All of our firewalls will trivially satisfy this notion. Note as well that we are not interested in protocols whose functionality "depends on the reverse firewall," so we require that the protocol without the reverse firewall is also functional.

Definition 1 (Reverse firewall). *A reverse firewall* $\mathcal{W}$ maintains functionality $\mathcal{F}$ for party A in protocol $\mathcal{P}$ *if protocol* $\mathcal{P}$ *satisfies* $\mathcal{F}$*, the protocol* $\mathcal{P}_{\mathsf{A}\to\mathcal{W}\circ\mathsf{A}}$ *satisfies* $\mathcal{F}$*, and the protocol* $\mathcal{P}_{\mathsf{A}\to\mathcal{W}\circ\cdots\circ\mathcal{W}\circ\mathsf{A}}$ *also satisfies* $\mathcal{F}$*. (I.e., we can compose arbitrarily many reverse firewalls without breaking functionality.)*

Of course, a firewall is not interesting unless it provides some benefit. The most natural reason to deploy a reverse firewall is to *preserve* the security of a protocol, even in the presence of compromise. The below definition (which again follows [MS14]) captures this notion by asking that the protocol obtained by replacing party A with $\mathcal{W} \circ \widetilde{A}$ for an arbitrary corrupted party $\widetilde{A}$ still achieves some notion of security. For example, when we consider message transmission, we will want the firewall to guarantee Alice's privacy against some adversary, even when Alice's own computer has been corrupted.

Definition 2 (Security preservation). *A reverse firewall* strongly preserves security $\mathcal{S}$ for party A in protocol $\mathcal{P}$ *if protocol* $\mathcal{P}$ *satisfies* $\mathcal{S}$*, and for any polynomial-time algorithm* $\bar{\mathsf{A}}$*, the protocol* $\mathcal{P}_{\mathsf{A}\to\mathcal{W}\circ\bar{\mathsf{A}}}$ *satisfies* $\mathcal{S}$*. (I.e., the firewall can guarantee security even when an adversary has tampered with A.)*

A reverse firewall preserves security $\mathcal{S}$ for party A in protocol $\mathcal{P}$ satisfying functionality requirements $\mathcal{F}$ *if protocol* $\mathcal{P}$ *satisfies* $\mathcal{S}$*, and for any polynomial-time algorithm* $\widetilde{\mathsf{A}}$ *such that* $\mathcal{P}_{\mathsf{A}\to\widetilde{\mathsf{A}}}$ *satisfies* $\mathcal{F}$*, the protocol* $\mathcal{P}_{\mathsf{A}\to\mathcal{W}\circ\widetilde{\mathsf{A}}}$ *satisfies* $\mathcal{S}$*. (I.e., the firewall can guarantee security even when an adversary has tampered with A, provided that the tampered implementation does not break the functionality of the protocol.)*

For technical reasons, we will also need a new definition not present in [MS14]. We wish to show composition theorems, allowing us to construct message-transmission protocols with secure reverse firewalls from key-agreement protocols with their own firewalls. In order to accomplish this, we will need the notion of *detectable failure*, which essentially just asks for an efficient algorithm that can distinguish between a transcript that could have been created by a valid run of the protocol and a transcript that must be invalid. We will use such an algorithm to make sure that a firewall of some larger message-transmission protocol can always

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

Fig. 1: $\mathsf{LEAK}(\mathcal{P}, \mathsf{A}, \mathsf{B}, \mathcal{W}, \lambda)$, the exfiltration resistance security game for a reverse firewall $\mathcal{W}$ for party A in protocol $\mathcal{P}$ against party B with input I. $S_{\bar{B}}$ is the state of $\bar{B}$ after the run of the protocol, and I may be any valid input for $\mathcal{P}$.

detect if the key-agreement sub-protocol has failed in some way. We make this precise below. In order to do so, we need to carefully consider what it means for a transcript to be valid. (The reader may wish to skip these definitions on a first read.)

Definition 3 (Valid transcripts). *A sequence of bits r* generates transcript $\mathcal{T}$ in protocol $\mathcal{P}$ *if there exists valid private input I such that a run of the protocol $\mathcal{P}$ with input I in which the parties' coin flips are taken from r results in the transcript $\mathcal{T}$. A transcript $\mathcal{T}$ is a* valid transcript for protocol $\mathcal{P}$ *if there is a sequence r generating $\mathcal{T}$ and no party outputs $\perp$ at the end of the protocol. (Here, we assume that the public input is part of the transcript.)*

Definition 4 (Detectable failure). *A reverse firewall $\mathcal{W}$* fails detectably for party A in protocol $\mathcal{P}$ *if*

- *if $\mathcal{T}$ is a valid transcript for $\mathcal{P}_{\mathsf{A} \to \mathcal{W} \circ \mathsf{A}}$ (i.e., if $\mathcal{T}$ could have been honestly generated by the protocol with the reverse firewall in place), then there is no randomness r generating $\mathcal{T}$ such that some party outputs $\perp$ at the end of the protocol run with coin flips r (i.e., a transcript's validity does not depend on coin flips or private input);*
- *the firewall $\mathcal{W}$ outputs the special symbol $\perp$ when run on any transcript that is not valid for $\mathcal{P}_{\mathsf{A} \to \mathcal{W} \circ \mathsf{A}}$; and*
- *there is a polynomial-time deterministic algorithm that decides whether a transcript $\mathcal{T}$ is valid for $\mathcal{W} \circ \mathsf{A}$ in $\mathcal{P}_{\mathsf{A} \to \mathcal{W} \circ \mathsf{A}}$.*

We will also need the notion of *exfiltration resistance*, introduced in [MS14]. We refer the reader to [MS14] for a much more thorough description of exfiltration resistance, but we provide a brief discussion below. Intuitively, a reverse firewall is exfiltration resistant if "no corrupt implementation of Alice can leak information through the firewall." We say that it is exfiltration resistant for Alice against Bob if Alice cannot leak information to Bob through the firewall, and we say that it is exfiltration resistant against eavesdroppers (or just exfiltration resistant) if Alice cannot leak information through the firewall to an adversary that is only given access to the protocol transcript.

The relationship between security preservation and exfiltration resistance depends on the security notion of the cryptographic primitive in question. E.g., in a message-transmission protocol, a reverse firewall for Alice resists exfiltration if and only if it preserves semantic security. However, it is possible to construct a reverse firewall for a key-agreement protocol that preserves security but does not resist exfiltration (for example, we can append an additional arbitrary message to the beginning of any key-agreement protocol without changing its security properties, but clearly such a message can be used to leak information), and it is also possible to construct such a firewall that resists exfiltration but does not preserve security. The second definition below (which uses the notion of valid transcripts) is new to this paper and is necessary for our composition theorems.

Definition 5 (Exfiltration resistance). *A reverse firewall is* exfiltration resistant for party A against party B in protocol $\mathcal{P}$ satisfying functionality $\mathcal{F}$ *if no PPT algorithm $\mathcal{E}$ with output circuits $\widetilde{\mathsf{A}}$ and $\widetilde{\mathsf{B}}$ such that $\mathcal{P}_{\mathsf{A} \to \widetilde{\mathsf{A}}}$ and $\mathcal{P}_{\mathsf{B} \to \widetilde{\mathsf{B}}}$ satisfy $\mathcal{F}$ has non-negligible advantage in $\mathsf{LEAK}(\mathcal{P}, \mathsf{A}, \mathsf{B}, \mathcal{W}, \lambda)$. If B is empty, then we simply say that the firewall is exfiltration resistant.*

A reverse firewall is exfiltration resistant for party A against party B in protocol $\mathcal{P}$ with valid transcripts *if no PPT algorithm $\mathcal{E}$ with output circuits $\widetilde{\mathsf{A}}$ and $\widetilde{\mathsf{B}}$ such that $\mathcal{P}_{\mathsf{A}\to\widetilde{\mathsf{A}}}$ and $\mathcal{P}_{\mathsf{B}\to\widetilde{\mathsf{B}}}$ produce valid transcripts for $\mathcal{P}$ has non-negligible advantage in $\mathsf{LEAK}(\mathcal{P},\mathsf{A},\mathsf{B},\mathcal{W},\lambda)$. If B is empty, then we simply say that the firewall is exfiltration resistant with valid transcripts.*

A reverse firewall is strongly exfiltration resistant for party A against party B in protocol $\mathcal{P}$ *if no PPT adversary $\mathcal{E}$ has non-negligible advantage in $\mathsf{LEAK}(\mathcal{P},\mathsf{A},\mathsf{B},\mathcal{W},\lambda)$. If B is empty, then we simply say that the protocol is strongly exfiltration resistant.*

For simplicity, we assume that honest parties always output $\perp$ when they receive a malformed message (e.g., when a message that should be a pair of group elements is not a pair of group elements).

2.2 Message-transmission protocols

A *message-transmission protocol* is a two-party protocol in which one party, Alice, is able to communicate a plaintext message to the other party, Bob. (For simplicity, we only formally model the case in which Alice wishes to send a single plaintext to Bob per run of the protocol, but this of course naturally extends to a more general case in which Alice and Bob wish to exchange many plaintext messages.) We consider two notions of security for such messages. First, we consider *CPA security*, in which the adversary must distinguish between the transcript of a run of the protocol in which Alice communicates the plaintext m_0 to Bob and the transcript with which Alice communicates m_1 to Bob, where m_0 and m_1 are adversarially chosen plaintexts. (Even in this setting, we allow the adversary to start many concurrent runs of the protocol with adaptively chosen plaintexts.) Our stronger notion of security is *CCA security* in which the adversary may "feed" the parties any messages and has access to a decryption oracle. Our security definitions are similar in spirit to [DF14], but adapted for our setting.

Session ids. Throughout this paper, we consider protocols that may be run concurrently many times between the same two parties. In order to distinguish one run of a protocol from another, we therefore "label" each run with a unique session id, denoted sid. We view sid as an implicit part of every message, and we often ignore sid when it is not important. Our parties and firewalls are stateful, and we assume that the parties and the firewall maintain a list of the relevant session ids, together with any information that is relevant to continue the run of the protocol corresponding to sid (such as the number of messages sent so far, any values that need to be used later in the protocol, etc.). We typically suppress explicit reference to these states. In our security games, the adversary may choose the value sid for each run of the protocol, provided that each party has a unique run for each session sid. (In fact, it does not even make sense for the adversary to use the same sid for two different runs of the protocol with the same party, as this party will necessarily view any calls with the same sid as corresponding to a single run of the protocol. However, as is clear from our security games, an active adversary may maintain two separate runs of a protocol with two different parties but the same sid.) In practice, sid can be a simple counter or any other nonce (perhaps together with any information necessary for communication, such as IP addresses). We note in passing that, in the setting of reverse firewalls, a counter is preferable to, e.g., a random nonce to avoid providing a channel through sid, but such concerns are outside the scope of this paper.

The definition below makes the above formal and provides some vocabulary.

Definition 6 (Message transmission protocol). *A* message-transmission protocol *is a two-party protocol in which one party, Alice, receives as input a plaintext m from some plaintext space $\mathcal{M}$. The protocol is* correct *if for any input $m \in \mathcal{M}$, Bob's output is always m.*

We represent the protocol by four algorithms $\mathcal{P} = (\mathsf{setup}, \mathsf{next}_{\mathsf{A}}, \mathsf{next}_{\mathsf{B}}, \mathsf{return}_{\mathsf{B}})$. setup takes as input 1^λ, where λ is the security parameter and returns the starting states for each party, $S_{\mathsf{A}}, S_{\mathsf{B}}$, which consist of both private input, σ_{A} and σ_{B} respectively, and public input π. Each party's next procedure is a stateful algorithm that takes as input sid and an incoming message; updates the party's state; and returns an outgoing message. The $\mathsf{return}_{\mathsf{B}}$ procedure takes as input Bob's state S_{B} and sid and returns Bob's final output.

We say that a message-transmission protocol is

- unkeyed *if setup does not return any private input σ_{A} or σ_{B};*
- singly keyed *if setup returns private input σ_{B} for Bob but none for Alice;*

– publicly keyed *if* setup *returns private input for both parties* σ_{A} *and* σ_{B}*, but these private inputs are independently distributed; and*
– privately keyed *if* setup *returns private input for both parties whose distributions are dependent.*

When we present protocols, we omit this formality, preferring instead to use diagrams as in Figure 4. But, this formulation is convenient for our security definitions. In particular, we present the relevant subprocedures for our security games in Figure 2. An adversary plays the game depicted in Figure 2 by first calling initialize (receiving as output π) and then making various calls to the other subprocedures. Each time it calls a subprocedure, it receives any output from the procedure. The game ends when the adversary calls finalize, and the adversary wins if and only if the output of finalize is 1.

proc. initialize(1^λ)
$(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}}, \pi) \xleftarrow{\$} \mathsf{setup}(1^\lambda)$
$S_{\mathsf{A}} \leftarrow (\sigma_{\mathsf{A}}, \pi)$; $S_{\mathsf{B}} \leftarrow (\sigma_{\mathsf{B}}, \pi)$
$\mathsf{sid}^* \leftarrow \perp$; compromised $\leftarrow$ false
$b \xleftarrow{\$} \{0,1\}$
`OUTPUT` π

proc. finalize(b^*)
`IF` $b = b^*$, `RETURN` 1
`ELSE`, `RETURN` 0

proc. start-run(sid, m)
`IF` $\mathsf{sid} \notin S_{\mathsf{A}}$, $S_{\mathsf{A}}.\mathsf{add}(\mathsf{sid}, m)$

proc. start-challenge(sid, m_0, m_1)
`IF` $\mathsf{sid} \notin S_{\mathsf{A}}$ `AND` $\mathsf{sid}^* = \perp$,
  $\mathsf{sid}^* \leftarrow \mathsf{sid}$
  $S_{\mathsf{A}}.\mathsf{add}(\mathsf{sid}, m_b)$

proc. get-next$_{\mathsf{A}}(\mathsf{sid}, M)$
`IF` compromised,
  `OUTPUT` $\perp$
`OUTPUT` $\mathsf{next}_{\mathsf{A}}(S_{\mathsf{A}}, \mathsf{sid}, M)$

proc. get-next$_{\mathsf{B}}(\mathsf{sid}, M)$
`IF` compromised,
  `OUTPUT` $\perp$
`OUTPUT` $\mathsf{next}_{\mathsf{B}}(S_{\mathsf{B}}, \mathsf{sid}, M)$

proc. get-output$_{\mathsf{B}}(\mathsf{sid})$
`IF` $\mathsf{sid} = \mathsf{sid}^*$ `OR` compromised,
  `OUTPUT` $\perp$
`OUTPUT` $\mathsf{return}_{\mathsf{B}}(S_{\mathsf{B}}, \mathsf{sid})$

proc. get-secrets
compromised $\leftarrow$ true
`OUTPUT` $(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}})$

Fig. 2: Procedures used to define security for message-transmission protocol $\mathcal{P} = (\mathsf{setup}, \mathsf{next}_{\mathsf{A}}, \mathsf{next}_{\mathsf{B}})$. An adversary plays this game by first calling initialize and then making various oracle calls. The game ends when the adversary calls finalize, and the output of finalize is one if the adversary wins and zero otherwise.

The below definitions capture formally the intuitive notions of security that we presented above. In particular, the CPA security definition allows the adversary to start arbitrarily many concurrent runs of the protocol with adversarial input, but it does not allow the adversary to change the messages sent by the two parties or to send its own messages. We also define forward secrecy, which requires that security hold even if the parties' secret keys may be leaked to the adversary.

Definition 7 (Message-transmission security). *A message-transmission protocol is called*

– chosen-plaintext secure *(CPA-secure) if no PPT adversary has non-negligible advantage in the game presented in Figure 2 when* get-next$_{\mathsf{A}}(\mathsf{sid}, M)$ *and* get-next$_{\mathsf{B}}(\mathsf{sid}, M)$ *output* $\perp$ *unless this is the first* get-next *call with this* sid *or* M *is the output from the previous* get-next$_{\mathsf{A}}$ *call with the same* sid *or the previous* get-next$_{\mathsf{B}}$ *with the same* sid *respectively (i.e., the adversary is passive); and*
– chosen-ciphertext secure *(CCA-secure) if no PPT adversary has non-negligible advantage in the game presented in Figure 2 with access to all oracles.*

We say that the protocol is chosen-plaintext (resp. chosen-ciphertext) secure without forward secrecy *if the above holds without access to the* get-secrets *oracle.*

We note that it does not make sense to consider chosen-ciphertext security when Bob may be corrupted. In this case, the output of get-output$_{\mathsf{B}}$ could be arbitrary. (Note that the firewall can potentially "sanitize" Bob's *messages*, but not his *output*.) We therefore only consider firewalls that preserve CPA security for Bob.

2.3 Key agreement

Key-agreement protocols will play a central role in our constructions, so we now provide a definition of key agreement that suffices for our purposes. Our notion of key agreement closely mirrors the definitions from the previous section.

Definition 8 (Key agreement). *A* key-agreement protocol *is represented by five algorithms,* $\mathcal{P} = (\mathsf{setup}, \mathsf{next}_\mathsf{A}, \mathsf{next}_\mathsf{B}, \mathsf{return}_\mathsf{A}, \mathsf{return}_\mathsf{B})$*.* setup *takes as input* 1^λ*, where* λ *is the security parameter and returns the starting states for each party,* $S_\mathsf{A}, S_\mathsf{B}$*, which consists of public input* π *and the private input for each party* σ_A *and* σ_B*. Each party's* next *procedure is a stateful algorithm that takes as input* sid *and an incoming message; updates the party's state; and returns an outgoing message. Each party's* return *procedure takes as input the relevant party's state and* sid *and returns the party's final output from some key space* $\mathcal{K}$ *or* $\perp$*. We also allow* auxiliary input aux *to be added to Alice's state before the first message of a protocol is sent.*

The protocol is correct *if Alice and Bob always output the same thing at the end of the run of a protocol for any random coins and auxiliary input* aux*.*

We say that a key-agreement protocol is

- unkeyed *if* setup *does not return any private input* σ_A *or* σ_B*;*
- singly keyed *if* setup *returns private input* σ_B *for Bob but no private input* σ_A *for Alice; and*
- publicly keyed *if* setup *returns private input for both parties* σ_A *and* σ_B*.*

proc. $\mathsf{initialize}(1^\lambda)$
$(\sigma_\mathsf{A}, \sigma_\mathsf{B}, \pi) \xleftarrow{\$} \mathsf{setup}(1^\lambda)$
$S_\mathsf{A} \leftarrow (\sigma_\mathsf{A}, \pi)$
$S_\mathsf{B} \leftarrow (\sigma_\mathsf{B}, \pi)$
$\mathsf{sid}^* \leftarrow \perp$
$\mathsf{compromised} \leftarrow \mathsf{false}$
$b \xleftarrow{\$} \{0, 1\}$
OUTPUT π

proc. $\mathsf{finalize}(b^*)$
IF $b = b^*$,
  RETURN 1
ELSE, RETURN 0

proc. $\mathsf{start\text{-}run}(\mathsf{sid}, \mathsf{aux})$
IF $\mathsf{sid} \notin S_\mathsf{A}$,
  $S_\mathsf{A}.\mathsf{add}(\mathsf{sid}, \mathsf{aux})$

proc. $\mathsf{start\text{-}challenge}(\mathsf{sid}, \mathsf{aux})$
IF $\mathsf{sid}^* = \perp$ AND $\mathsf{sid} \notin S_\mathsf{A}$,
  $\mathsf{sid}^* \leftarrow \mathsf{sid}$
  $\mathcal{R}_{\mathsf{sid}^*} \xleftarrow{\$} \mathcal{K}$
  $S_\mathsf{A}.\mathsf{add}(\mathsf{sid}, \mathsf{aux})$

proc. $\mathsf{get\text{-}next}_\mathsf{A}(\mathsf{sid}, M)$
IF NOT $\mathsf{compromised}$,
  OUTPUT $\mathsf{next}_\mathsf{A}(S_\mathsf{A}, \mathsf{sid}, M)$

proc. $\mathsf{get\text{-}next}_\mathsf{B}(\mathsf{sid}, M)$
IF NOT $\mathsf{compromised}$,
  OUTPUT $\mathsf{next}_\mathsf{B}(S_\mathsf{B}, \mathsf{sid}, M)$

proc. $\mathsf{get\text{-}output}_\mathsf{A}(\mathsf{sid})$
IF $\mathsf{compromised}$, OUTPUT $\perp$
IF $\mathsf{sid} = \mathsf{sid}^*$ AND $b = 0$,
  IF $\mathsf{return}_\mathsf{A}(S_\mathsf{A}, \mathsf{sid}) = \perp$, OUTPUT $\perp$
  ELSE, OUTPUT R_sid
ELSE, OUTPUT $\mathsf{return}_\mathsf{A}(S_\mathsf{A}, \mathsf{sid})$

proc. $\mathsf{get\text{-}output}_\mathsf{B}(\mathsf{sid})$
IF $\mathsf{compromised}$, OUTPUT $\perp$
IF $\mathsf{sid} = \mathsf{sid}^*$ AND $b = 0$,
  IF $\mathsf{return}_\mathsf{B}(S_\mathsf{B}, \mathsf{sid}) = \perp$, OUTPUT $\perp$
  ELSE, OUTPUT R_sid
ELSE, OUTPUT $\mathsf{return}_\mathsf{B}(S_\mathsf{B}, \mathsf{sid})$

proc. $\mathsf{get\text{-}secrets}$
$\mathsf{compromised} \leftarrow \mathsf{true}$
OUTPUT $(\sigma_\mathsf{A}, \sigma_\mathsf{B})$

Fig. 3: Procedures used to define security for key-agreement protocol $\mathcal{P} = (\mathsf{setup}, \mathsf{next}_\mathsf{A}, \mathsf{next}_\mathsf{B}, \mathsf{return}_\mathsf{A}, \mathsf{return}_\mathsf{B})$. An adversary plays this game by first calling $\mathsf{initialize}$ and then making various oracle calls. The game ends when the adversary calls $\mathsf{finalize}$, and the output of $\mathsf{finalize}$ is one if the adversary wins and zero otherwise. We suppress the auxiliary input aux when it is irrelevant.

Definition 9 (Key-agreement security). *A key-agreement protocol is*

- secure against passive adversaries *if no probabilistic polynomial-time adversary has non-negligible advantage in the game presented in Figure 3 when* $\mathsf{get\text{-}next}_\mathsf{A}(\mathsf{sid}, M)$ *and* $\mathsf{get\text{-}next}_\mathsf{B}(\mathsf{sid}, M)$ *output* $\perp$ *unless this is the first* $\mathsf{get\text{-}next}$ *call with this* sid *or* M *is the output from the previous* $\mathsf{get\text{-}next}_\mathsf{B}$ *call with the same* sid *or the previous* $\mathsf{get\text{-}next}_\mathsf{A}$ *call with the same* sid *respectively (i.e., the adversary is passive);*
- secure against active adversaries for Alice *if no probabilistic polynomial-time algorithm has non-negligible advantage in the game presented in Figure 3 without access to the* $\mathsf{get\text{-}output}_\mathsf{A}$ *oracle;*
- secure against active adversaries for Bob *if no probabilistic polynomial-time algorithm has non-negligible advantage in the game presented in Figure 3 without access to the* $\mathsf{get\text{-}output}_\mathsf{B}$ *oracle; and*
- secure against active adversaries *if it is secure against active adversaries for both Bob and Alice; and*
- authenticated for Bob *if no probabilistic polynomial-time algorithm playing the game presented in Figure 3 can output a valid transcript with corresponding session id* sid *unless* $\mathsf{return}_\mathsf{B}(S_\mathsf{B}, \mathsf{sid}) \neq \perp$ *or* $\mathsf{compromised} = \mathsf{true}$*. (I.e., it is hard to find a valid transcript unless Bob returns a key.) Furthermore, if the transcript is valid and* $\mathsf{get\text{-}output}_\mathsf{A}(\mathsf{sid}) \neq \perp$ *then* $\mathsf{get\text{-}output}_\mathsf{A}(\mathsf{sid}) = \mathsf{return}_\mathsf{B}(\mathsf{sid})$*. (I.e., if the transcript is valid and Alice outputs a key, then Bob outputs the same key.)*

Note that these definitions are far from standard. In particular, in the case of active adversaries, we define security for Alice in terms of the keys that *Bob* outputs and security for Bob in terms of the keys that *Alice* outputs. This may seem quite counterintuitive. But, in our setting, we are worried that Alice may be corrupted. In this case, we cannot hope to restrict Alice's output after she receives invalid messages. (The firewall can modify Alice's *messages*, but not her *output*.) So, the best we can hope for is that the firewall prevents a tampered implementation of Alice (together with an active adversary) from "tricking" Bob into returning an insecure key.

3 The unkeyed setting

In this section we investigate the scenario in which Alice and Bob have neither a shared secret key nor each other's public keys. Note that we of course can only achieve CPA security here because neither Alice nor Bob can be authenticated. As such, this section's primary purpose is to illustrate core concepts and to act as a useful stepping stone toward the more complicated and stronger protocols that we present in Section 4. Two approaches are feasible: a simple two-round solution in which Bob (the receiver) generates a public-private key pair and sends the public key to Alice, and a more flexible and efficient protocol based on key agreement.

3.1 A two-round protocol from rerandomizable encryption

We first consider the simple case of CPA-secure two-round schemes in which the first message is a public key chosen randomly by Bob and the second message is an encryption of Alice's plaintext under this public key. Figure 4 shows the protocol.

Alice		**Bob**
		$(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}$
	$\xleftarrow{\quad pk \quad}$	
	$\xrightarrow{\quad c := \mathsf{Enc}_{pk}(m) \quad}$	
		$\mathtt{OUTPUT}\ \mathsf{Dec}_{sk}(c)$

Fig. 4: Two round message-transmission protocol using the public-key encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.

In order to provide a reverse firewall for Alice in this protocol, the encryption scheme must be *rerandomizable*. In order to provide a reverse firewall for Bob, the scheme must be *key malleable*. Intuitively, a scheme is key malleable if a third party can "rerandomize" a public key and map ciphertexts under the "rerandomized" public key to ciphertexts under the original public key. We provide formal definitions below.

Definition 10 (Public-key encryption). *A* public-key encryption scheme *is a triple of algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. KeyGen *takes as input* 1^λ *where* λ *is the security parameter and outputs a public-key/private-key pair,* (pk, sk). Enc *takes as input the public key and a plaintext* m *from some plaintext space* $\mathcal{M}$ *with* $|\mathcal{M}| = 2^\lambda$ *and outputs a ciphertext* c *from some ciphertext space* $\mathcal{C}$. Dec *takes as input a ciphertext and the private key and outputs a plaintext or the special symbol* $\perp$. *We sometimes omit the keys from the input to* Enc *and* Dec. *The scheme is* correct *if* $\mathsf{Dec}(\mathsf{Enc}(m)) = m$ *for all* $m \in \mathcal{M}$. *The scheme is* semantically secure *if for any adversarially chosen pair of plaintexts* (m_0, m_1), $\mathsf{Enc}(m_0)$ *is computationally indistinguishable from* $\mathsf{Enc}(m_1)$.

Definition 11 (Rerandomizable encryption). *A semantically secure public-key encryption scheme is* rerandomizable *if there is an efficient algorithm* Rerand *(with access to the public key) such that for any ciphertext* c *such that* $\mathsf{Dec}(c) \neq \perp$, *we have* $\mathsf{Rerand}(\mathsf{Dec}(c)) = \mathsf{Dec}(c)$, *and the pair* $(c, \mathsf{Rerand}(c))$ *is computationally indistinguishable from* $(c, \mathsf{Rerand}(\mathsf{Enc}(0)))$. *We say that it is* strongly rerandomizable *if it is rerandomizable and for any string* c *(not necessarily a valid ciphertext),* $(c, \mathsf{Rerand}(c))$ *is computationally indistinguishable from* $(c, \mathsf{Rerand}(\mathsf{Enc}(0)))$.

Definition 12 (Key malleability). *A public-key encryption scheme is* key malleable *if the output of* KeyGen *is distributed uniformly over the space of valid keys, for each public key pk there is a unique associated private key sk, and there is a pair of efficient algorithms* $\mathsf{KeyMaul}$ *and* $\mathsf{CKeyMaul}$ *that behave as follows.* $\mathsf{KeyMaul}$ *is a randomized algorithm that takes as input a public key pk and returns a new public key* pk' *whose distribution is uniformly random over the public key space and independent of pk. Let* (sk, pk) *be a private key/public key pair. Let* (sk', pk') *be the unique pair associated with randomness r such that* $pk' = \mathsf{KeyMaul}(pk; r)$*. Then,* $\mathsf{CKeyMaul}$ *takes as input a ciphertext c and randomness r and returns* c' *such that* $\mathsf{Dec}_{sk'}(c) = \mathsf{Dec}_{sk}(c')$*. We suppress the input r when it is understood. Furthermore, we require that* $\mathsf{KeyMaul}$ *outputs a uniformly random key* pk' *if called on input that is not in the public-key space.*

Example. It is well-known that ElGamal encryption [ElG85] is both key malleable and strongly rerandomizable. In particular, given an ElGamal public-key (g, h) over a group of order p, a ciphertext (u, v) can be rerandomized by applying the operation $(u, v) \to (g^r u, h^r v)$ where $r \xleftarrow{\$} \mathbb{Z}_p^*$ is chosen uniformly at random. The public key can be mauled by applying the operation $(g, h) \to (g^\alpha, h^\beta)$ where $(\alpha, \beta) \xleftarrow{\$} (\mathbb{Z}_p^*)^2$ are chosen uniformly and independently at random. Finally, a ciphertext (u, v) under key (g^α, h^β) can be converted into a ciphertext under (g, h) by applying the operation $(u, v) \to (u^{\beta/\alpha}, v)$.

If the underlying encryption scheme in Figure 4 is rerandomizable, then we can build a reverse firewall for Alice as in Figure 5. If it is key malleable, then we can build a reverse firewall for Bob as in Figure 6. The following theorem shows that this protocol and its reverse firewalls are secure.

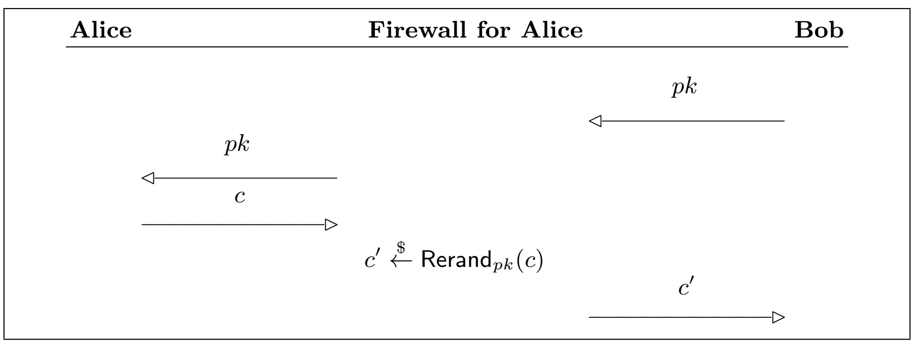

Fig. 5: Reverse firewall for Alice for the protocol shown in Figure 5 that works if the encryption scheme is rerandomizable.

Theorem 1. *The unkeyed message-transmission protocol shown in Figure 4 is CPA-secure if the underlying encryption scheme is semantically secure. If the scheme is also rerandomizable, then the reverse firewall shown in Figure 5 maintains functionality and preserves security for Alice and resists exfiltration for Alice. If the scheme is strongly rerandomizable, then this reverse firewall strongly preserves security for Alice and strongly resists exfiltration for Alice. If the scheme is key malleable, then the reverse firewall shown in Figure 6 maintains functionality for Bob, strongly preserves Bob's security, and strongly resists exfiltration for Bob against Alice.*

Proof. It is a common folklore result that the underlying protocol (i.e., the protocol without reverse firewalls) is CPA-secure. It is clear that the two firewalls maintain functionality.

Security and exfiltration resistance of Bob's firewall follows from the definition of key malleability. In particular, for any tampered implementation $\bar{B}$ of Bob, after the post-processing by the reverse firewall, the key pk' is uniformly random by the definition of key malleability, regardless of the behavior of $\bar{B}$. This implies exfiltration resistance, and security then follows from the fact that the underlying protocol is secure when the key is chosen legitimately.

Consider a tampered implementation of Alice $\widetilde{\mathsf{A}}$ that maintains functionality, and let c be the output of Alice. Since Alice maintains functionality, $\mathsf{Dec}(c) \neq \perp$. So, by the definition of rerandomizability, the

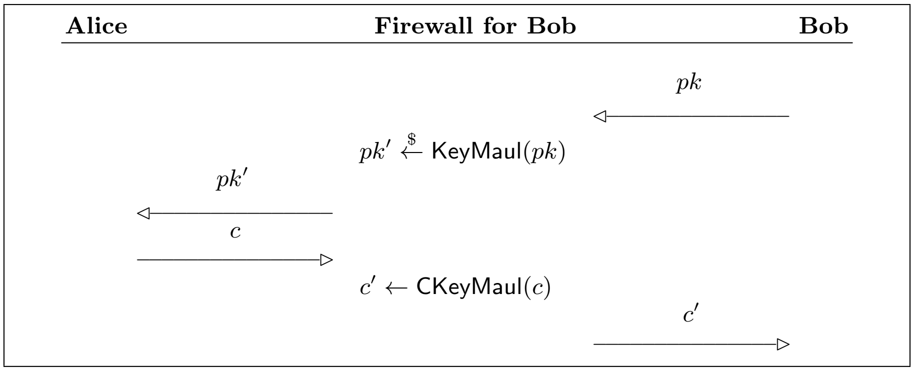

Fig. 6: Reverse firewall for Bob for the protocol shown in Figure 4 that works if the encryption scheme is key malleable. We suppress the randomness r used as input to $\mathsf{KeyMaul}$ and $\mathsf{CKeyMaul}$.

output of $\mathsf{Rerand}(c)$ is indistinguishable from $\mathsf{Enc}_{pk}(0)$. The security preservation and exfiltration resistance of Alice's firewall follows. A similar argument shows that strong rerandomizability implies strong security preservation and strong exfiltration resistance. $\square$

Hybrid encryption fails. A major drawback of the above scheme is that it requires public-key operations of potentially very long plaintexts, which can be very inefficient in practice. A common solution is to use *hybrid encryption*, in which $\mathsf{Enc}_{pk}(m)$ is replaced by $(\mathsf{Enc}_{pk}(rk), \mathsf{SEnc}_{rk}(m))$, where SEnc is some suitable symmetric-key encryption scheme and rk is a freshly chosen uniformly random key for SEnc. However, if we simply replace the public-key encryption in Figure 4 with the corresponding hybrid-key encryption scheme, then this fails spectacularly. In one attack, a tampered version of Alice $\widetilde{\mathsf{A}}$ can choose some *fixed* secret key rk^* and send the message $(\mathsf{Enc}_{pk}(rk^*), \mathsf{SEnc}_{rk^*}(m))$. If rk^* is a valid key, then $\widetilde{\mathsf{A}}$ maintains functionality, but an adversary that knows rk^* can of course read any messages that Alice sends.

3.2 A solution using key agreement

Recall that we are in the unkeyed setting, so we are still interested in CPA security. (We address CCA security in the next section.) The protocol from the previous section works, but it requires a public-key operation on the plaintext, which may be very long. In practice, this can be very inefficient. Above, we show that one common solution to this problem in the classical setting, hybrid encryption, fails with reverse firewalls because it allows Alice to *choose* a key rk that will be used to encrypt the plaintext—thus allowing a tampered version of Alice to "choose a bad key."

So, we instead consider an alternative common solution to this efficiency problem: key agreement followed by symmetric-key encryption. (See Figure 7.) As in Appendix A, we use a nonce-based encryption scheme with unique ciphertexts. We can view this as a modification of hybrid encryption in which "Alice and Bob together choose the key rk" that will be used to encrypt the plaintext. More importantly from our perspective, the messages that define the key will go through the firewall. As an added benefit, once a key is established, Alice can use it to efficiently send multiple messages, not just one, without any additional public-key operations (though we do not model this here). The composition theorem below shows that this protocol can in fact have a reverse firewall for both parties, provided that the key-agreement protocol itself has a reverse firewall that satisfies some suitable security requirements.[4] See Appendix B for the proof. In the next section, we construct such a protocol.

[4] The conditions of our theorem actually suffice to provide security in the stronger model in which Alice may send many messages per run of the protocol. To keep our definitions relatively simple, we do not model this formally and prove the weaker statement. We note in passing that the weaker statement actually only requires semantically secure symmetric-key encryption. The same is true of Theorem 4.

$$
\begin{array}{lcr}
\textbf{Alice} & & \textbf{Bob} \\
\hline
& m_1 := \mathsf{next_A}(\mathsf{sid}) & \\
& \xrightarrow{\hspace{10em}} & \\
& m_2 := \mathsf{next_B}(\mathsf{sid}, m_2) & \\
& \xleftarrow{\hspace{10em}} & \\
& \vdots & \\
& m_\ell := \mathsf{next_B}(\mathsf{sid}, m_{\ell-1}) & \\
& \xleftarrow{\hspace{10em}} & \\
sk_\mathsf{A} \leftarrow \mathsf{return_A}(\mathsf{sid}) & & \\
c \leftarrow \mathsf{Enc}_{sk_\mathsf{A}}(\mathsf{sid}, m) & c & \\
& \xrightarrow{\hspace{10em}} & \\
& & sk_\mathsf{B} \leftarrow \mathsf{return_B}(\mathsf{sid}) \\
& & \mathtt{OUTPUT}\ \mathsf{Dec}_{sk_\mathsf{B}}(\mathsf{sid}, c)
\end{array}
$$

Fig. 7: The message-transfer protocol obtained by combining a key-agreement scheme (setup, $\mathsf{next_A}$, $\mathsf{next_B}$, $\mathsf{return_A}$, $\mathsf{return_B}$) and a nonce-based encryption scheme, $(\mathsf{Enc}, \mathsf{Dec})$.

Theorem 2 (Composition theorem for CPA security). *Let $\mathcal{W}_\mathsf{A}$ and $\mathcal{W}_\mathsf{B}$ be reverse firewalls in the underlying key-agreement protocol in Figure 7 for Alice and Bob respectively. Let $\mathcal{W}_\mathsf{A}^*$ be the firewall for Alice in the full protocol in Figure 7 obtained by applying $\mathcal{W}_\mathsf{A}$ to the key-agreement messages and then letting the last message through if $\mathcal{W}_\mathsf{A}$ does not output $\perp$ and replacing the last message by $\perp$ otherwise. Let $\mathcal{W}_\mathsf{B}^*$ be the firewall for Bob in the full protocol in Figure 7 obtained by applying $\mathcal{W}_\mathsf{B}$ ot the key-agreement messages and simply letting the last message through. Then,*

1. *the protocol in Figure 7 is CPA-secure if the underlying key-agreement protocol is secure against passive adversaries and the underlying nonce-based encryption scheme is CPA-secure;*
2. *$\mathcal{W}_\mathsf{B}^*$ preserves CPA security if $\mathcal{W}_\mathsf{B}$ preserves security of the key-agreement protocol; and*
3. *$\mathcal{W}_\mathsf{A}^*$ preserves CPA security if the encryption scheme has unique ciphertexts and $\mathcal{W}_\mathsf{A}$ preserves semantic security and is exfiltration resistant against Bob.*

Finally, we note that strong security preservation is not possible for this protocol (at least for Alice).

Remark 1 (Informal). There is no reverse firewall for Alice in the protocol illustrated in Figure 7 that maintains functionality and strongly preserves Alice's security.

Proof. Consider the tampered implementation of Alice $\bar{\mathsf{A}}$ that goes through the key-agreement protocol as normal and then sends as its last message $c := \mathsf{Enc}_{sk'}(\mathsf{sid}, m)$ for some fixed key sk' chosen by the adversary by simulating a run of the key-agreement protocol. Since the firewall maintains functionality, it must be the case that the message sent by the firewall to Bob c' satisfies $\mathsf{Dec}_{sk'}(\mathsf{sid}, c') = m$. So, the adversary can decrypt the message itself. Clearly, this protocol is not secure (by any reasonable definition). $\square$

3.3 Key agreement secure against passive adversaries

Theorem 2 motivates the study of unkeyed key-agreement protocols with reverse firewalls that preserve security against passive adversaries. In the classical setting (i.e., without reverse firewalls), the canonical example is the elegant key-agreement protocol of Diffie and Hellman [DH06], shown in Figure 8, whose security follows immediately from the hardness of DDH over the base group G. We use this as an example to illustrate the basic idea of a reverse firewall in the key-agreement setting.

Alice		**Bob**
$a \xleftarrow{\$} \mathbb{Z}_p^*$		
	$\xrightarrow{A := g^a}$	
		$b \xleftarrow{\$} \mathbb{Z}_p^*$
	$\xleftarrow{B := g^b}$	
`OUTPUT` B^a		`OUTPUT` A^b

Fig. 8: Diffie-Hellman key agreement over a group G of prime order p with generator g.

Diffie-Hellman key agreement has a simple reverse firewall for Alice, which raises both messages to a single random power, $\alpha \in \mathbb{Z}_p^*$. We present this reverse firewall in Figure 9. Note that this firewall effectively replaces Alice's message with a uniformly random message. Security then follows from the security of the underlying protocol, since the transcript and resulting key in the two cases are distributed identically. This very basic idea of rerandomizing Diffie-Hellman key agreement is behind all of our protocols in this section.

Alice		**Alice's Firewall**		**Bob**
	$\xrightarrow{A}$			
		`IF` $A \notin G \setminus \{1_G\}$, $\quad A \xleftarrow{\$} \mathbb{Z}_p^*$ $\alpha \xleftarrow{\$} \mathbb{Z}_p^*$		
			$\xrightarrow{A^\alpha}$	
			$\xleftarrow{B}$	
	$\xleftarrow{B^\alpha}$			

Fig. 9: Reverse firewall for Alice in the protocol from Figure 8.

But, this protocol cannot have a reverse firewall that maintains correctness and preserves security for Bob. Consider the tampered implementation $\widetilde{B}$ that repeatedly samples $b \xleftarrow{\$} \mathbb{Z}_p^*$ and computes A^b until the first bit of the result is zero. $\widetilde{B}$ then sends g^b to Alice as normal. If the firewall maintains correctness, then Alice's output must be A^b, but this output is clearly distinguishable from random—its first bit is zero! (Note that this attack is very similar to the attack on hybrid encryption that we discussed at the bottom of Section 3.1.) This is a problem that we must overcome if we want to build protocols that have secure firewalls for *both* parties. In particular, we run the risk that one party has the ability to "accept or reject" any key.

To solve this problem, we add an additional message to the beginning of the protocol in which Bob commits to the message that he will send later. Of course, in order to permit a secure firewall, the commitment scheme itself must be both malleable (so that the firewall can rerandomize the underlying message that Bob has committed to, mapping a commitment of B to a commitment of B^α) and rerandomizable (so that the randomness used by Bob to commit and open will not leak any information about his message). To achieve our strongest level of security, we also need the scheme to be statistically hiding and for each commitment to have a unique opening for a given message. (These requirements are easily met in practice. For example, a simple variant of the Pedersen commitment suffices [Ped92]. For completeness, we present such a scheme in Appendix C.) The protocol is shown in Figure 10. In Figure 11, we present a single reverse firewall for this protocol that happens to work for either party. (Each party would need to deploy its own version of this firewall to guarantee its own security. It just happens that each party's firewall would have the same "code.")

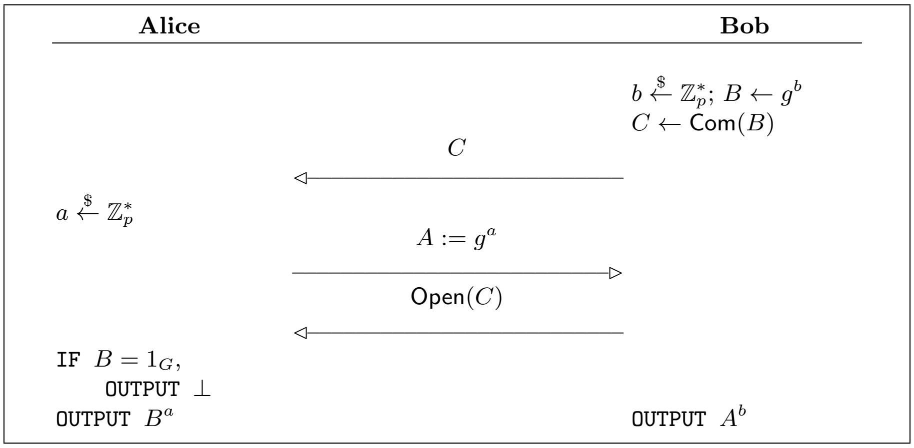

Fig. 10: A variant of Diffie-Hellman key agreement over a group G of prime order p with public generator g. $(\mathsf{Com}, \mathsf{Open})$ is a commitment scheme.

Theorem 3. *The protocol in Figure 10 is secure against passive adversaries if DDH is hard in G. The reverse firewall $\mathcal{W}$ in Figure 11 is functionality maintaining. If the commitment scheme is statistically hiding, then $\mathcal{W}$ preserves security for Alice and is strongly exfiltration resistant against Bob. If the commitment scheme is computationally binding, then $\mathcal{W}$ is exfiltration resistant for Bob against Alice and preserves security for Bob. $\mathcal{W}$ also fails detectably for both parties.*

Proof. It is clear that the underlying protocol is secure provided that DDH is hard in G. It is also clear that $\mathcal{W}$ maintains functionality and fails detectably for both parties. (Here, we assume that the firewall outputs $\perp$ if it ever receives a malformed message.)

Note that, after rerandomization and mauling, the commitment C' is a uniformly random commitment of a uniformly random group element, independent of the original commitment C. Since Bob is functionality maintaining, his second message is fixed unless he can find an alternative opening for the commitment, which by assumption is computationally hard. It follows that $\mathcal{W}$ is exfiltration resistant for Bob against Alice and preserves security for Bob.

To prove strong exfiltration resistance for Alice against Bob and security preservation, we again note that Bob's first message is a uniformly random commitment of a uniformly random group element. Since the commitment is statistically binding, it is statistically close to independent from α, regardless of Bob's choice of C. Therefore, Alice's message A is statistically close to independent from α, and A^α is statistically close to uniform. The result follows. $\square$

4 The publicly keyed setting

We now consider the publicly keyed setting. I.e., we assume that both Bob and Alice have a public key and a private key (though their private keys must be independent). In the setting of the previous section, with no public-key infrastructure, it is trivially impossible to achieve CCA-security. (An adversary can simply "pretend to be Bob" and read Alice's plaintext.) In this section, we show that a CCA-secure message-transmission protocol with reverse firewalls does in fact exist in the publicly keyed setting. In particular, in Section 4.1, we give the CCA analogue of Theorem 2, showing that a key-agreement protocol that is secure against active adversaries and has sufficiently secure reverse firewalls together with a symmetric-key

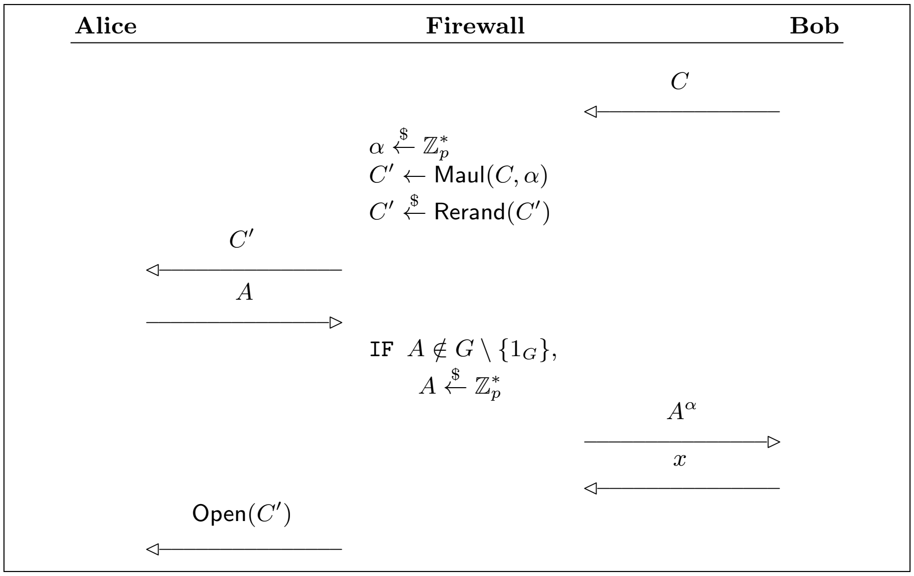

Fig. 11: Reverse firewall for either Alice or Bob in the protocol from Figure 8. $\mathsf{Maul}(C, \alpha)$ takes a commitment $C = \mathsf{Com}(B)$ and converts it into a commitment of B^α. $\mathsf{Rerand}(C)$ takes a commitment $C = \mathsf{Com}(B)$ and converts it into a uniformly random commitment of B. We assume that a rerandomized and mauled commitment can be opened with access to an opening of the original commitment.

encryption scheme suffices. As in the previous section, this key-agreement-based protocol has the additional benefit that it is efficient, in the sense that it does not apply public-key operations to the plaintext. In Section 4.2, we construct a key-agreement protocol that suffices.

4.1 A solution using key agreement

The theorem below shows that we can build an efficient CCA-secure message-transmission protocol with reverse firewalls in this setting, provided that we have a sufficiently secure key-agreement protocol with reverse firewalls. (Recall that Bob's reverse firewall can only preserve CPA security. Such a firewall is already given by Theorem 2, so we do not repeat this here.) See Appendix D for the proof.

Theorem 4 (Composition theorem for CCA security). *Define $\mathcal{W}_\mathsf{A}$ and $\mathcal{W}_\mathsf{A}^*$ as in Theorem 2. Then,*

1. *the protocol in Figure 7 is CCA-secure if the underlying key-agreement protocol is secure against active adversaries for Alice and the underlying nonce-based encryption scheme is CCA-secure; and*
2. *$\mathcal{W}_\mathsf{A}^*$ preserves CCA-security if the encryption scheme has unique ciphertexts, the key-agreement protocol is authenticated for Bob, and $\mathcal{W}_\mathsf{A}$ preserves security for Alice, is exfiltration resistant against Bob with valid transcripts, and fails detectably.*

4.2 Key agreement secure against active adversaries

Theorem 4 motivates the study of key-agreement protocols with reverse firewalls that preserve security against active adversaries. In the classical setting, the common solution is essentially for each of the parties to sign the transcript of this run of the protocol. Intuitively, this lets Alice know that she sees the same transcript as Bob, and vice versa. However, this solution does not work in our setting. In particular, it is important for us that messages *can* be altered without breaking functionality, so that the firewall can rerandomize messages when necessary.

Of course, while we want to allow for the possibility that Alice and Bob disagree on the transcript but still output a key, we do want them to agree on the key itself. This leads to the idea of signing some deterministic function of the key, so that the signatures can be used to verify that the parties share the same key without necessarily requiring them to share the same transcript. This is the heart of our solution.

However, we also have to worry that the signatures themselves can provide channels, allowing tampered versions of the parties to leak some information. We solve this by using a unique signature scheme, as defined by [MRV99]. These guarantee that there is a unique signature that verifies for each plaintext, so that functionality-maintaining implementations of Alice and Bob have only one option for their signature, given a fixed key. (See [AMV15] for a thorough analysis of signatures in the context of reverse firewalls and corrupted implementations, including alternative ways to implement signatures that would suffice for our purposes.)

Furthermore, in order for our firewall to fail detectably, it has to be able to check the signature itself—so that it can distinguish a valid transcript from an invalid one. So, we would like the parties to sign a deterministic function of g^{ab} that is efficiently computable given only access to g^a and g^b. This leads naturally to the use of a symmetric bilinear map $e : G \times G \to G_T$. The parties then sign $e(g^a, g^b)$. Of course, g^{ab} is no longer indistinguishable from random in the presence of a bilinear map. But, it can be hard to compute. So, we apply a cryptographic hash function H to g^{ab} in order to extract the final key $H(g^{ab})$.

We now provide two definitions to make this precise.

Definition 13 (Unique Signatures). *A* unique signature scheme *is a triple of algorithms* $(\mathsf{KeyGen}, \mathsf{USig}, \mathsf{Ver})$. KeyGen *takes as input* 1^λ *where* λ *is the security parameter and outputs a public key* pk *and a private key* sk. Sig *takes as input the secret key* sk *and a plaintext* m *and outputs a signature* τ. Ver *takes as input the public key* pk*, a signature* τ *and a message* m *and outputs either* true *or* false. *A signature scheme is* correct *if* $\mathsf{Ver}_{pk}(\mathsf{USig}_{sk}(m), m) = \mathsf{true}$. *It is* unique *if for each plaintext* m *and public key* pk*, there is a unique signature* τ *such that* $\mathsf{Ver}_{pk}(\tau, m) = \mathsf{true}$.

A signature scheme is secure against adaptive chosen-message existential-forgery attacks *if no adversary with access to the public key and a signature oracle can produce a valid signature not returned by the oracle.*

We will need to use a group with a symmetric bilinear map in which the following variant of the computational Diffie-Hellman assumption holds.

Definition 14 (Inverse CDH). *For a group* G *of order* p*, we say that* inverse CDH is hard in G *if no probabilistic polynomial-time adversary taking input* (g, g^a, g^b) *where* $g \xleftarrow{\$} G$ *and* $(a, b) \xleftarrow{\$} \mathbb{Z}_p^2$ *has non-negligible probability of returning* $(h, h^{1/a}, h^{1/b})$ *for some element* $h \in G \setminus \{1_G\}$.

Note that inverse CDH is stronger than the standard CDH assumption, as (g^{ab}, g^b, g^a) is a solution to inverse CDH.

We now present our protocol in Figure 12 with a reverse firewall for both parties in Figure 13. It requires a unique signature scheme $(\mathsf{USig}, \mathsf{Ver})$ with public keys pk_A for Alice and pk_B for Bob and corresponding secret keys sk_A and sk_B respectively, a base group G with generator g in which inverse CDH is hard, a target group G_T, and a non-trivial bilinear map between the two groups $e : G \times G \to G_T$. We also need a function $H : G \to \{0,1\}^\ell$ for some polynomially large ℓ that extracts hardcore bits from CDH. Presumably a standard cryptographic hash function will work. For simplicity, we model H as a random oracle, but we note that the proof can be modified to apply to any function H such that $(g^a, g^b, H(g^{ab}))$ is indistinguishable from random (see Kiltz for candidate hash functions [Kil07]). We stress again that this protocol is remarkably efficient, and we think that it can and should be used in practice.

Theorem 5. *The protocol shown in Figure 12 is authenticated for Bob and secure against active adversaries if the signature scheme is secure and inverse CDH is hard in* G*. The reverse firewall* $\mathcal{W}$ *shown in Figure 13 preserves security against active adversaries for Alice, preserves authenticity, is exfiltration resistant for Alice against Bob with valid transcripts, and fails detectably for Alice.* $\mathcal{W}$ *also preserves security against active adversaries for Bob, is exfiltration resistant for Bob against Alice with valid transcripts, and fails detectably for Bob.*

Proof. The fact that the protocol is authenticated for Bob is clear. Informally, any valid transcript must have a valid signature from Bob. By the security of the signature scheme, except with negligible probability, the adversary cannot produce a signature with a given sid without Bob returning a key. The same proof

Alice		**Bob**
`INPUT:` sk_{A}	`INPUT:` $(pk_{\mathsf{A}}, pk_{\mathsf{B}}, \mathsf{sid})$	`INPUT:` sk_{B}
$a \xleftarrow{\$} \mathbb{Z}_p^*$; $A \leftarrow g^a$ $C \leftarrow \mathsf{Com}(A)$		
	$\xrightarrow{\quad C \quad}$	
		$b \xleftarrow{\$} \mathbb{Z}_p^*$
	$\xleftarrow{\quad B := g^b \quad}$	
`IF` $B = 1_G$, `OUTPUT` $\perp$ $k \leftarrow e(A, B)$ $\tau_{\mathsf{A}} \leftarrow \mathsf{USig}_{sk_{\mathsf{A}}}(k, \mathsf{sid})$		
	$\xrightarrow{\quad (\mathsf{Open}(C), \tau_{\mathsf{A}}) \quad}$	

		`IF` $A = 1_G$,
		    `OUTPUT` $\perp$
		$k \leftarrow e(A, B)$
		`IF NOT` $\mathsf{Ver}(\tau_{\mathsf{A}}, k, \mathsf{sid})$,
		    `OUTPUT` $\perp$
		$\tau_{\mathsf{B}} \leftarrow \mathsf{USig}_{sk_{\mathsf{B}}}(k, \mathsf{sid})$

	$\xleftarrow{\quad \tau_{\mathsf{B}} \quad}$	
`IF NOT` $\mathsf{Ver}(\tau_{\mathsf{B}}, k, \mathsf{sid})$, `OUTPUT` $\perp$ `OUTPUT` $H(B^a)$		`OUTPUT` $H(A^b)$

Fig. 12: Authenticated key agreement with a firewall for both parties. USig is a unique signature.

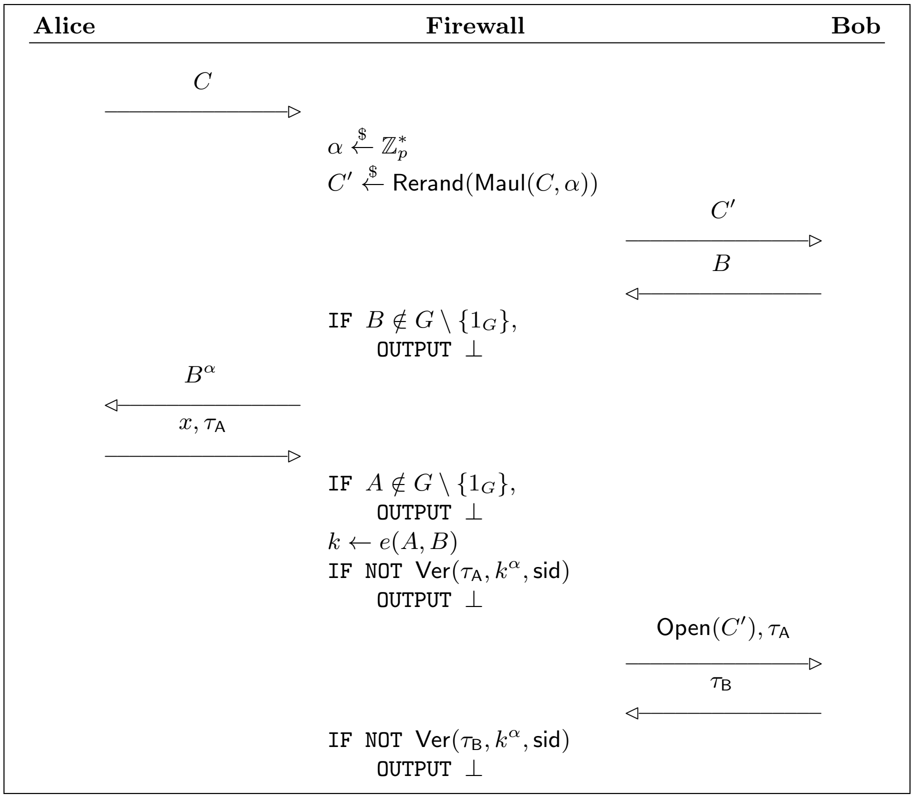

Fig. 13: Reverse firewall for either Alice or Bob in the protocol from Figure 12. C is a commitment of the group element A. $\mathsf{Maul}(C, \alpha)$ takes a commitment $C = \mathsf{Com}(A)$ and converts it into a commitment of A^α. $\mathsf{Rerand}(C)$ takes a commitment $C = \mathsf{Com}(A)$ and converts it into a uniformly random commitment of A. We assume that a rerandomized and mauled commitment can be opened with access to an opening of the original commitment.

shows that the firewall preserves authenticity. The fact that the firewall fails detectably for both parties is also clear.

We prove that $\mathcal{W}$ preserves security against active adversaries for Alice. Let $\widetilde{\mathsf{A}}$ be a functionality-maintaining tampered implementation of Alice. Consider the following sequence of games.

- **Game** 1 is the key-agreement security game against active adversaries for Alice in the protocol shown in Figure 12 with A replaced by $\mathcal{W} \circ \widetilde{\mathsf{A}}$.
- **Game** 2 is **Game** 1 in which the adversary never provides a signature τ_A or τ_B unless it came from a response from Alice or Bob respectively.
- **Game** 3 is **Game** 2 in which Alice's commitment C is replaced by A (the value that her commitment opens to), and the firewall's commitment C' is replaced by A^α in all runs of the protocol. We also remove Alice's second message is simply removed and join Bob's last two messages into a single message, (B, τ_B).
- **Game** 4 is **Game** 3 in which Alice's message A is replaced by a uniformly random group element, and the firewall takes $\alpha := 1$ (i.e., the firewall leaves all messages alone, besides checking signatures) in all runs of the protocol.

The following claim follows from the definition of adaptive chosen-message existential-forgery security.

Claim 5.1. *If the signature scheme is secure, then for any PPT adversary $\mathcal{E}$, $|\mathsf{Adv}^{\textbf{\textit{Game}}\ 1}(\mathcal{E}) - \mathsf{Adv}^{\textbf{\textit{Game}}\ 2}(\mathcal{E})|$ is negligible.*

It is trivial to take any adversary in **Game** 2 and convert it to an adversary in **Game** 3 with the same advantage. Similarly, **Game** 4 only differs from **Game** 3 syntactically. Indeed, $A' := A^\alpha$ is a uniformly random group element independent of everything else, and $B' := B^\alpha$ is uniformly random and independent of A' and everything else. (Recall that Bob is honest.) So, since Alice's next message is deterministic (because the signature is unique and Alice maintains functionality) and no output from Alice is given to the adversary other than these two messages (recall that the $\mathsf{get\text{-}output_A}$ oracle is not allowed in this game), this is merely a change of variables.

Claim 5.2. *No PPT adversary $\mathcal{E}$ has non-negligible advantage in* **Game** 4 *if inverse CDH is hard in G.*

Proof. We assume without loss of generality that $\mathcal{E}$ makes no "trivial" oracle calls, whose output can be easily predicted based on prior calls, such as calls to $\mathsf{get\text{-}output_B}$ before the relevant run of the protocol is finished. We build an adversary $\mathcal{E}'$ in the inverse CDH game as follows. On input (g_1, g_2, g_3), $\mathcal{E}'$ first generates the keys for the signature scheme, $(pk_\mathsf{A}, pk_\mathsf{B}, sk_\mathsf{A}, sk_\mathsf{B})$, and passes $(g_1, pk_\mathsf{A}, pk_\mathsf{B})$ to $\mathcal{E}$. As $\mathcal{E}'$ will be simulating many runs of the protocol defined by **Game** 4, for convenience, we assume that $\mathcal{E}$ "oracle" calls to the protocol as in the key-agreement security game. $\mathcal{E}'$ simply passes the random oracle calls of $\mathcal{E}$ to its own random oracle (or simulates a random oracle), keeping a list of all calls. Finally, $\mathcal{E}'$ responds to other oracle calls of $\mathcal{E}$ as follows.

- When $\mathcal{E}$ calls $\mathsf{start\text{-}run}(\mathsf{sid})$, $\mathcal{E}'$ calls its own "oracle" $\mathsf{start\text{-}run}(\mathsf{sid})$.
- When $\mathcal{E}$ calls $\mathsf{start\text{-}challenge}(\mathsf{sid}^*)$, $\mathcal{E}'$ stores sid^*.
- When $\mathcal{E}$ calls $\mathsf{get\text{-}next_A}(\mathsf{sid}, M)$, if $\mathsf{sid} \neq \mathsf{sid}^*$, $\mathcal{E}'$ calls its own "oracle" $\mathsf{get\text{-}next_A}(\mathsf{sid}, M)$ and passes the response to $\mathcal{E}$. Otherwise, it sets $h_1 \leftarrow M$ and passes $(g_3, \mathsf{Sig}_{sk}(e(h_2, g_3), \mathsf{sid})$ to $\mathcal{E}$.
- When $\mathcal{E}$ calls $\mathsf{get\text{-}next_B}(\mathsf{sid}, M)$, if $\mathsf{sid} \neq \mathsf{sid}^*$, $\mathcal{E}'$ calls its own "oracle" $\mathsf{get\text{-}next_B}(\mathsf{sid}, M)$ and passes the response to $\mathcal{E}$. Otherwise, if this is the first message of the run, it passes g_2 to $\mathcal{E}$. Otherwise, it sets $(\tau, h_1) \leftarrow M$.
- When $\mathcal{E}$ calls $\mathsf{get\text{-}output_B}(\mathsf{sid})$, if $\mathsf{sid} \neq \mathsf{sid}^*$, $\mathcal{E}'$ calls its own "oracle" $\mathsf{get\text{-}output_B}(\mathsf{sid})$. If $\mathsf{sid} = \mathsf{sid}^*$, and $e(h_1, g_3) \neq e(g_2, h_2)$, then it returns $\perp$. Otherwise, $\mathcal{E}'$ responds with a uniformly random string.
- When $\mathcal{E}$ calls $\mathsf{get\text{-}secrets}$, $\mathcal{E}'$ responds with sk.
- When $\mathcal{E}$ calls $\mathsf{finalize}(b^*)$, $\mathcal{E}'$ stops simulating and proceeds as below.

Note that the view of $\mathcal{E}$ is identical to its view in **Game** 4 unless it calls the random oracle on the actual key, $h_3 := \mathsf{DH}(h_1, g_3) = \mathsf{DH}(g_2, h_2)$. Note as well that, because a random oracle is extractable, if $\mathcal{E}$ has non-negligible advantage, it must call the random oracle on the key. Therefore, $\mathcal{E}'$ can search through its random oracle queries until it finds h_3 satisfying $e(g_1, h_3) = e(h_1, g_3)$. It then returns (h_3, h_2, h_1). If it finds nothing, it returns $\perp$. The result follows from noting that (h_3, h_2, h_1) is a valid solution to CDH. (5.2) ■

It follows that $\mathcal{W}$ preserves security against active adversaries for Alice. The proof for Bob is essentially identical. And, essentially the same proof shows exfiltration resistance. □

5 The singly keyed setting

Finally, we consider the singly keyed setting, in which Bob has a public-key/private-key pair, but Alice does not. As in the classical case, we show that there exist one-round singly keyed protocols that are not forward secret. These are essentially the single-round analogue of the two-round protocol presented in Figure 4 in Section 3.1. We show that the existence of such a protocol is equivalent to the existence of rerandomizable encryption, and we show how to achieve CCA-security (though not forward secrecy).

5.1 One-round CPA-secure protocols

The next theorem shows that one-round CPA-secure protocols with reverse firewalls are equivalent to rerandomizable public-key encryption.

Theorem 6. *Any rerandomizable (resp. strongly rerandomizable) semantically secure public-key encryption scheme implies a one-round CPA-secure singly keyed message-transmission protocol without forward security with a reverse firewall that preserves security (resp. strongly preserves security) and resists exfiltration (resp. strongly resists exfiltration). Conversely, any one-round CPA-secure message-transmission protocol with a reverse firewall that preserves security (resp. strongly preserves security) implies a rerandomizable (resp. strongly rerandomizable) semantically secure public-key encryption scheme.*

Proof. To prove the first statement, we consider the protocol in which Alice simply sends Bob an encryption of the plaintext under Bob's public key. Alice's firewall applies the Rerand algorithm to the plaintext. (Bob does not need a firewall, since he does not send any messages.) Security of this protocol follows immediately from the security of the encryption scheme, and the fact that the firewall preserves security follows immediately from the rerandomizability of the encryption scheme.

To prove the second statement, consider the following encryption scheme. The key generation algorithm runs the setup algorithm of the underlying MTP protocol, receiving as output σ_{A}, σ_{B}, and π. The public key is then π and σ_{A}, and the private key is σ_{B}. The encryption algorithm first uses σ_{A} and an arbitrarily chosen sid to compute Alice's single message in the protocol, given the plaintext. It then applies the reverse firewall to this message; the result is the ciphertext. The rerandomization algorithm simply applies the reverse firewall to this message. The security and rerandomizability of the scheme are immediate from the security of the underlying MTP and the security of the firewall respectively. □

5.2 A one-round CCA-secure protocol

To extend this idea to stronger notions of security, we need the underlying encryption scheme to satisfy stronger notions of security. A natural candidate is CCA security. However, CCA-secure encryption schemes cannot be rerandomizable, so we need a slightly weaker notion. RCCA security, as defined by [CKN03], suffices, and rerandomizable RCCA-secure schemes do exist (see, e.g., [Gro04,PR07]), though they are relatively inefficient. We present the RCCA security game in Figure 14. In addition, we need a rerandomized ciphertext to be indistinguishable from a valid encryption "even with access to a decryption oracle." Figure 15 and the definition below makes this precise.

proc. $\mathsf{IND\text{-}RCCA}(\lambda)$
$(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$(m_0, m_1) \leftarrow \mathcal{E}^{\mathcal{O}_1}(pk)$
$b \xleftarrow{\$} \{0,1\}$; $C^* \xleftarrow{\$} \mathsf{Enc}_{pk}(m_b)$
$b^* \leftarrow \mathcal{E}^{\mathcal{O}_2}(\sigma, c^*)$
`OUTPUT` $(b = b^*)$

proc. $\mathcal{O}_1(c)$
`OUTPUT` $\mathsf{Dec}_{sk}(c)$

proc. $\mathcal{O}_2(c)$
$m \leftarrow \mathsf{Dec}_{sk}(c)$
`IF` $m = m_0$ `OR` $m = m_1$,
    `OUTPUT` Challenge
`ELSE`,
    `OUTPUT` m

Fig. 14: The RCCA security game.

proc. $\mathsf{IND\text{-}RCCA}(\lambda)$
$(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$(c_0, c_1) \leftarrow \mathcal{E}^{\mathcal{O}_1}(pk)$
$b \xleftarrow{\$} \{0,1\}$; $c^* \xleftarrow{\$} \mathsf{Rerand}_{pk}(c_b)$
$b^* \leftarrow \mathcal{E}^{\mathcal{O}_2}(c^*)$
`OUTPUT` $(b = b^*)$

proc. $\mathcal{O}_1(c)$
`OUTPUT` $\mathsf{Dec}_{sk}(c)$

proc. $\mathcal{O}_2(c)$
$m \leftarrow \mathsf{Dec}_{sk}(c)$
`IF` $m = \mathsf{Dec}_{sk}(c_0)$ `OR` $m = \mathsf{Dec}_{sk}(c_1)$,
    `OUTPUT` Challenge
`ELSE`,
    `OUTPUT` m

Fig. 15: The RCCA rerandomization game.

Definition 15. *An encryption scheme is* RCCA secure *if no probabilistic polynomial-time adversary $\mathcal{E}$ has non-negligible advantage in the game presented in Figure 14. It is* RCCA rerandomizable *if there exists an algorithm* Rerand *with access to the public key such that for any ciphertext c with* $\mathsf{Dec}(c) \neq \perp$, $\mathsf{Dec}(\mathsf{Rerand}(c)) = \mathsf{Dec}(c)$ *and no probabilistic polynomial-time adversary $\mathcal{E}$ has non-negligible advantage in the game presented in Figure 15 when we require that* $\mathsf{Dec}(c_i) \neq \perp$. *It is* strongly RCCA-rerandomizable *if the previous statement holds even if* $\mathsf{Dec}(c_i) = \perp$.

The below theorem is the CCA analogue of Theorem 6.

Theorem 7. *Any RCCA-rerandomizable (resp. strongly rerandomizable), RCCA-secure encryption scheme implies a one-round CCA-secure singly keyed message-transmission protocol without forward security with a reverse firewall that preserves security (resp. strongly preserves security) and resists exfiltration (resp. strongly resists exfiltration).*

Proof. Consider the protocol in which Alice, given as input a plaintext m and a session id sid, simply sends the message $\mathsf{Enc}_{pk}(\mathsf{sid}, m)$. On input c, Bob's return function computes $(\mathsf{sid}^\dagger, m) \leftarrow \mathsf{Dec}_{sk}(c)$. If $\mathsf{sid}^\dagger = \mathsf{sid}$, it outputs m. Otherwise, it outputs $\perp$. Alice's firewall simply applies the Rerand algorithm to Alice's message.

For any PPT adversary $\mathcal{E}$ in the CCA-security game against this message-transmission protocol (without forward security), we construct $\mathcal{E}'$ in the RCCA-security game against the underlying encryption scheme with $\mathsf{Adv}(\mathcal{E}') = \mathsf{Adv}(\mathcal{E})$. We assume without loss of generality that $\mathcal{E}$ never makes a "useless" oracle call. I.e., $\mathcal{E}$ never calls $\mathsf{start\text{-}run}$ with an already used sid, never calls $\mathsf{get\text{-}next}_\mathsf{A}$ with a never used sid, never calls $\mathsf{get\text{-}output}$ on the challenge sid, etc. The adversary $\mathcal{E}'$ behaves as follows in response to the oracle calls of $\mathcal{E}$.

- When $\mathcal{E}$ calls the $\mathsf{start\text{-}run}$ oracle with input (sid, m), $\mathcal{E}'$ sets $c_{\mathsf{sid,A}} \leftarrow \mathsf{Enc}_{pk}(\mathsf{sid}, m)$.
- When $\mathcal{E}$ calls $\mathsf{start\text{-}challenge}(\mathsf{sid}, m_0, m_1)$, $\mathcal{E}'$ sets $\mathsf{sid}^* \leftarrow \mathsf{sid}$. It then returns (sid^*, m_0^*) and (sid^*, m_1^*) as its challenge plaintexts, receiving in response the challenge ciphertext c^*. It sets $c_{\mathsf{sid}^*} = c^*$.
- When $\mathcal{E}$ calls $\mathsf{get\text{-}next}_\mathsf{A}(\mathsf{sid})$, $\mathcal{E}'$ replies with $c_{\mathsf{sid,A}}$.
- When $\mathcal{E}$ calls $\mathsf{get\text{-}next}_\mathsf{B}(\mathsf{sid}, c)$, $\mathcal{E}'$ sets $c_{\mathsf{sid,B}} \leftarrow c$.
- When $\mathcal{E}$ calls $\mathsf{get\text{-}output}_\mathsf{B}(\mathsf{sid})$, $\mathcal{E}'$ calls $\mathcal{O}_1(c_{\mathsf{sid,B}})$. If the output is not of the form (sid, m), $\mathcal{E}'$ responds with $\perp$. Otherwise, it responds with m.
- When $\mathcal{E}$ calls $\mathsf{finalize}(b^*)$, $\mathcal{E}'$ simply returns b^*.

Note that the view of $\mathcal{E}$ is identical to its view in the CCA-security game against the message-transmission protocol. Furthermore, $\mathcal{E}'$ wins the RCCA-security game if and only if $\mathcal{E}$ wins its simulated game. The result follows.

It follows that the protocol is CCA secure. The proof that the firewall preserves security is nearly identical to the above proof. ☐

5.3 Achieving forward secrecy and efficiency?

Note that the protocols described above suffer from two problems: they do not have forward secrecy, and they are inefficient (i.e., they require public-key operations on the entire plaintext). Ideally, we would like a composition theorem in this setting to match Theorems 2 and 4. Such a theorem would solve both problems, allowing us to achieve an efficient CCA-secure and forward-secret message-transmission protocol in the singly keyed setting. But, we are so far unable to prove such a theorem. So, we leave this as an open question.

As a potential alternative direction to achieving forward secrecy, we note that the protocol from Theorem 7 can be converted into a two-round CCA- secure and forward-secret protocol with a reverse firewall for Alice *but no reverse firewall for Bob*. In particular, in the first round of the protocol, Bob generates a fresh pair of keys $(pk^\dagger, sk^\dagger)$ for an RCCA-secure RCCA-rerandomizable encryption scheme and sends Alice $pk^\dagger$ together with a signature τ of $(\mathsf{sid}, pk^\dagger)$, where the signature is under Bob's signature key. Alice checks the signature and, if it is valid, sends Bob an encryption of her plaintext under the secret key. This is essentially the CCA analogue of the protocol in Section 3.1. As in that case, Alice's reverse firewall can simply check the signature and rerandomize the ciphertext. This protocol is in fact CCA secure, and Alice's firewall does preserve this security. However, we do not know how to construct a reverse firewall for Bob in this setting. In analogy with Section 3.1, a possible method would be to find an RCCA-rerandomizable encryption scheme that is also key malleable. We know of no such scheme, but even if such a scheme were constructed, the signature scheme would have to be similarly malleable, while still achieving an appropriate notion of security. So, we also leave this as an open question.

6 Conclusion and open questions

We consider the problem of message-transmission protocols in the cryptographic reverse firewalls framework of [MS15]. We show that this problem has a rich structure, in analogy with the classical setting, with a variety of solutions that require different setup assumptions, achieve different levels of security, and provide different levels of efficiency. Perhaps surprisingly, we show that it is possible to achieve concurrent, interactive CCA security in the presence of compromise against functionality-maintaining adversaries and CPA security against arbitrary adversaries. Many of our protocols (including those that provide the strongest notions of security) are very efficient and relatively simple, so that they can (and should) be implemented in practice.

Therefore, the most important work that we leave open is the implementation of our protocols. The most important theoretical question that we leave open is whether there is a non-trivial composition theorem in the singly keyed case, in analogy with Theorem 4. Note that Theorem 2 naturally extends to the singly keyed case, but we see no inherent reason why CCA-security should not be achievable from key agreement followed by symmetric-key encryption in this setting.

In addition, our work brings new attention to the question of rerandomizable RCCA-secure schemes. In particular, in Appendix 5, we show that such schemes give a one-round CCA-secure message-transmission protocol with a reverse firewall (without forward security). However, we do not know of such schemes that are "strongly rerandomizable" (as defined in Section 5). If such schemes existed, then we show that they would immediately imply one-round CCA-secure message transmission with a *strongly secure* reverse firewall. This would be the first such construction of this primitive. Similarly, our scheme in Section 3.1 shows the benefits of key-malleability in the reverse firewalls setting. Together with the results of Appendix 5, this leads naturally to the question of whether or not key-malleable rerandomizable RCCA-secure encryption exists.

References

[AMV15] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. Cryptology ePrint Archive, Report 2015/517, 2015. http://eprint.iacr.org/.

[BBG13] James Ball, Julian Borger, and Glenn Greenwald. Revealed: how US and UK spy agencies defeat internet privacy and security. *Guardian Weekly*, September 2013.

[BPR14a] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In *CRYPTO 2014*, volume 8616 of *LNCS*, pages 1–19. Springer, 2014. Full version: [BPR14b].

[BPR14b] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. Cryptology ePrint Archive, Report 2014/438, 2014. http://eprint.iacr.org/.

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *LNCS*, pages 565–582. Springer, 2003.

[CVE14a] Vulnerability summary for CVE-2014-1260 ('Heartbleed'). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1260, April 2014.

[CVE14b] Vulnerability summary for CVE-2014-1266 ('goto fail'). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1266, February 2014.

[CVE14c] Vulnerability summary for CVE-2014-6271 ('Shellshock'). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271, September 2014.

[DF14] Yevgeniy Dodis and Dario Fiore. Interactive encryption and message authentication. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks*, volume 8642 of *Lecture Notes in Computer Science*, pages 494–513. Springer International Publishing, 2014.

[DGG+15] Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 101–126. Springer Berlin Heidelberg, 2015.

[DH06] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006.

[DPSW06] Yvo Desmedt, Josef Pieprzyk, Ron Steinfeld, and Huaxiong Wang. A non-malleable group key exchange protocol robust against active insiders. In *Information Security*, volume 4176 of *Lecture Notes in Computer Science*, pages 459–475. Springer Berlin Heidelberg, 2006.

[ElG85] Taher ElGamal. 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.

[Gre14] Glenn Greenwald. *No Place to Hide: Edward Snowden, the NSA, and the U.S. Surveillance State.* Metropolitan Books, May 2014.

[Gro04] Jens Groth. Rerandomizable and replayable adaptive chosen ciphertext attack secure cryptosystems. In Moni Naor, editor, *TCC*, volume 2951 of *LNCS*, pages 152–170. Springer, 2004.

[Kil07] Eike Kiltz. Chosen-ciphertext secure key-encapsulation based on gap hashed Diffie-Hellman. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, volume 4450 of *Lecture Notes in Computer Science*, pages 282–297. Springer, 2007.

[KS05] Jonathan Katz and Ji Sun Shin. Modeling insider attacks on group key-exchange protocols. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, pages 180–189, New York, NY, USA, 2005. ACM.

[LHA+12] Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter. Public keys. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology — CRYPTO 2012*, volume 7417 of *LNCS*, pages 626–642. Springer Berlin Heidelberg, 2012.

[MRV99] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (New York, 1999)*, pages 120–130. IEEE Computer Soc., Los Alamitos, CA, 1999.

[MS14] Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. Cryptology ePrint Archive, Report 2014/758, 2014. `http://eprint.iacr.org/`.

[MS15] Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In *EUROCRYPT 2015*, 2015. Available from [MS14].

[Ped92] TorbenPryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology ?Äî CRYPTO ?Äô91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Berlin Heidelberg, 1992.

[PLS13] Nicole Perlroth, Jeff Larson, and Scott Shane. N.S.A. able to foil basic safeguards of privacy on Web. *The New York Times*, September 2013.

[PR07] Manoj Prabhakaran and Mike Rosulek. Rerandomizable RCCA encryption. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 517–534. Springer, 2007.

[PW03] Josef Pieprzyk and Huaxiong Wang. Key control in multi-party key agreement protocols. Workshop on Coding, Cryptography and Combinatorics, 2003.

[Sim84] Gustavus Simmons. The prisoners' problem and the subliminal channel. In David Chaum, editor, *Advances in Cryptology*, pages 51–67. Springer US, 1984.

[Sup15] `https://www.us-cert.gov/ncas/alerts/TA15-051A`, February 2015.

A The symmetric-key setting

As a warmup, we first consider the setting in which Alice and Bob share a private key. We observe that an elegant one-round protocol due to Bellare, Paterson, and Rogaway provides a solution that does not even need a reverse firewall [BPR14a]. We will also use this scheme in the sequel to build protocols that do not rely on shared private keys. We first define nonce-based encryption.

Definition 16 (Nonce-based encryption). *A* nonce-based symmetric-key encryption scheme *is a pair of deterministic algorithms* $(\mathsf{Enc}, \mathsf{Dec})$*.* Enc *takes as input a key from a key space* $\mathcal{K}$*, a nonce from a nonce space* $\mathcal{N}$*, and a plaintext from a plaintext space* $\mathcal{M}$ *and outputs a ciphertext from a ciphertext space* $\mathcal{C}$*.* Dec *takes as input a key, a nonce, and a ciphertext and returns a plaintext or the special symbol* $\perp$*. The scheme is* correct *if for any key sk, nonce r, and plaintext m,* $\mathsf{Dec}(r, \mathsf{Enc}_{sk}(r, m)) = m$*.*

Such a scheme is CPA secure *if no probabilistic polynomial-time adversary can distinguish between* $\mathsf{Enc}_{sk}(r^*, m_0)$ *and* $\mathsf{Enc}_{sk}(r^*, m_1)$ *with non-negligible advantage where* r^**,* m_0*, and* m_1 *are adversarially chosen when given access to an encryption oracle that outputs* $\perp$ *unless given a unique nonce* r*. It is* CCA-secure *if no probabilistic polynomial-time has non-negligible advantage when also given access to a decryption oracle that outputs* $\perp$ *if* $r = r^*$*.*

Such a scheme scheme $(\mathsf{Enc}, \mathsf{Dec})$ has unique ciphertexts *if for any key sk, message m, and nonce r, there is exactly one ciphertext c such that* $\mathsf{Dec}(r, c) = m$*.*

Theorem 8. *Let* $(\mathsf{Enc}, \mathsf{Dec})$ *be a nonce-based symmetric-key encryption scheme. Then, if the encryption scheme is CPA-secure (resp. CCA-secure) the one-round protocol in which Alice sends* $\mathsf{Enc}_{sk}(\mathsf{sid}, m)$ *and Bob returns* $\mathsf{Dec}_{sk}(\mathsf{sid}, m)$ *is a CPA-secure (resp. CCA-secure) one-round privately keyed message-transmission* *protocol without forward secrecy. If the encryption scheme has unique ciphertexts, then the "trivial" reverse firewall that simply passes Alice's messages to Bob unchanged preserves security and is exfiltration resistant against Bob.*

See, e.g., [BPR14a] for formal analysis and the construction of such a scheme. The key thing to note from our perspective is that, as Bellare et al. observe, the fact that the encryption scheme has unique ciphertexts implies that any tampered version of Alice that maintains functionality necessarily behaves identically to honest Alice. The next theorem shows that we essentially cannot do any better for a one-round protocol without using public-key primitives. The proof is in appendix E.

Theorem 9. *There is a black-box reduction from semantically secure public-key encryption to CPA-secure symmetric-key encryption with at least four possible plaintexts and a reverse firewall that strongly preserves CPA security.*

Of course, the primary drawbacks of this approach are that it requires Alice and Bob to share a secret key and that it does not offer forward secrecy.

B Proof of Theorem 2

Proof. The security of the underlying protocol (i.e., without firewalls) follows by a folklore composition theorem. We assume that the last message of the key-agreement protocol is sent by Bob. (This is without loss of generality, as we can always add an empty message from Bob at the end of the protocol.) We prove Item 2. The proof of Item 3 is a simplified version of the proof of Theorem 4.

Let $\widetilde{B}$ be some functionality-maintaining tampered implementation of Bob. Let $\widetilde{\mathsf{B}}^*$ be the tampered implementation of Bob in the key-agreement protocol obtained by simply "truncating" $\widetilde{\mathsf{B}}$. (This is merely a syntactic change.) We assume without loss of generality that the adversary makes no "trivial" calls whose output can be trivially predicted from its previous oracle calls. E.g., it makes no calls to $\mathsf{get\text{-}output}$ (which always either returns $\perp$ or the plaintext that the adversary provided for the corresponding sid in the CPA-security game), no $\mathsf{get\text{-}next}$ calls with modified messages, etc. Consider the following sequence of games.

- **Game** 1 is the CPA-security game against the message-transfer protocol with Bob replaced by $\mathcal{W}^*_{\mathsf{B}} \circ \widetilde{\mathsf{B}}$.
- **Game** 2 is **Game** 1 in which the final message of the challenge run of the protocol is replaced by an encryption of the challenge plaintext m_b under a uniformly random key sk.
- **Game** 3 is **Game** 2 in which the final message of the challenge run is replaced by an encryption of 0 under a uniformly random key.

Note that no adversary can have non-zero advantage in the last game, as none of the messages depend on the challenge bit.

Claim 2.1. *If the encryption scheme has unique ciphertexts, the key-agreement protocol is secure against passive adversaries, and $\mathcal{W}_{\mathsf{B}}$ preserves this security, then for any PPT adversary $\mathcal{E}$, $|\mathsf{Adv}^{\textbf{\textit{Game}}\ 1}(\mathcal{E}) - \mathsf{Adv}^{\textbf{\textit{Game}}\ 2}(\mathcal{E})|$ is negligible.*

Proof. We construct a passive adversary $\mathcal{E}'$ in the security game against the key-agreement protocol with Bob replaced by $\mathcal{W}_{\mathsf{B}} \circ \widetilde{\mathsf{B}}^*$ as follows. $\mathcal{E}'$ receives as input the public parameters π and passes them to $\mathcal{E}$. $\mathcal{E}'$ then selects $b^\dagger \xleftarrow{\$} \{0,1\}$ uniformly at random and sets $S^\dagger_{\mathsf{A}} \leftarrow \emptyset$. It responds to the (non-trivial) oracle calls of $\mathcal{E}$ as follows.

- When $\mathcal{E}$ calls $\mathsf{start\text{-}run}(\mathsf{sid}, m)$, $\mathcal{E}'$ adds (sid, m) to $S^\dagger_{\mathsf{A}}$ and calls its own oracle $\mathsf{start\text{-}run}(\mathsf{sid})$.
- When $\mathcal{E}$ calls $\mathsf{start\text{-}challenge}(\mathsf{sid}^*, m_0, m_1)$, $\mathcal{E}'$ adds $(\mathsf{sid}^*, m_{b^\dagger})$ to $S^\dagger_{\mathsf{A}}$ and calls its own oracle $\mathsf{start\text{-}challenge}(\mathsf{sid}^*)$.
- When $\mathcal{E}$ calls $\mathsf{get\text{-}next}_{\mathsf{A}}(\mathsf{sid}, M)$, $\mathcal{E}'$ calls its own oracle $\mathsf{get\text{-}next}_{\mathsf{A}}(\mathsf{sid}, M)$. If this is not the last message of the protocol, $\mathcal{E}'$ then simply passes the resulting message to $\mathcal{E}$. If it is the last message, it $\mathcal{E}'$ calls its own oracle $\mathsf{get\text{-}output}_{\mathsf{A}}(\mathsf{sid})$, receiving as output some key sk. It responds with $\mathsf{Enc}_{sk}(\mathsf{sid}, m)$ where m is the unique plaintext such that $(\mathsf{sid}, m) \in S^\dagger_{\mathsf{A}}$. (Note that since the oracle calls of $\mathcal{E}$ are non-trivial, it must have made a unique call to either $\mathsf{start\text{-}run}$ or $\mathsf{start\text{-}challenge}$ with this sid.)

– When $\mathcal{E}$ calls $\mathsf{get\text{-}next}_{\mathsf{B}}(\mathsf{sid}, M)$, if this is not the last message of the protocol, $\mathcal{E}'$ calls its own oracle $\mathsf{get\text{-}next}_{\mathsf{B}}(\mathsf{sid}, M)$ and passes the resulting message to $\mathcal{E}$. Otherwise, $\mathcal{E}'$ does nothing.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}secrets}$, $\mathcal{E}'$ calls its own oracle $\mathsf{get\text{-}secrets}$ and passes the result to $\mathcal{E}$.
– When $\mathcal{E}$ calls $\mathsf{finalize}(b^*)$, $\mathcal{E}'$ returns 1 if $b^\dagger = b^*$ and 0 otherwise.

Suppose the challenge bit b in the key-agreement security game is 0, so that the challenge key sk^* corresponding to the challenge session id sid^* was selected uniformly at random. Then, clearly the view of $\mathcal{E}$ is identical to its view in **Game** 2, and $\mathcal{E}'$ correctly outputs 0 if and only if $\mathcal{E}$ "loses" its simulated game by returning $b^\dagger = b^*$. If, on the other hand, the challenge bit b is 1, then the view of $\mathcal{E}$ is identical to its view in **Game** 1, and $\mathcal{E}'$ correctly outputs 1 if and only if $\mathcal{E}$ "wins" its simulated game. The result follows. (2.1) ■

Claim 2.2. *If the encryption scheme is CPA-secure, then for any PPT adversary $\mathcal{E}$, $|\mathsf{Adv}^{\textbf{\textit{Game}}\ 2}(\mathcal{E}) - \mathsf{Adv}^{\textbf{\textit{Game}}\ 3}(\mathcal{E})|$ is negligible.*

Proof. We construct an adversary $\mathcal{E}'$ in the CPA-security game against the encryption scheme as follows. $\mathcal{E}'$ first runs the setup procedure of the key-agreement protocol, receiving as output σ_{A}, σ_{B}, and π. $\mathcal{E}'$ will simulate many runs of the key-agreement protocol with Bob replaced by $\mathcal{W}_{\mathsf{B}} \circ \mathsf{B}^*$ and input $(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}}, \pi)$. For convenience, we give $\mathcal{E}'$ an "oracle interface" to these simulated runs with "oracle" calls $\mathsf{start\text{-}run}$, $\mathsf{get\text{-}next}_{\mathsf{A}}$, $\mathsf{get\text{-}next}_{\mathsf{B}}$, and $\mathsf{get\text{-}output}_{\mathsf{A}}$ as in the key-agreement security game (Figure 3). E' sends π to $\mathcal{E}$, selects $b^\dagger \xleftarrow{\$} \{0, 1\}$ uniformly at random, and sets $S^\dagger_{\mathsf{A}} \leftarrow \emptyset$. It then responds to the oracle queries of $\mathcal{E}$ as follows.

– When $\mathcal{E}$ calls $\mathsf{start\text{-}run}(\mathsf{sid}, m)$, $\mathcal{E}'$ adds (sid, m) to $S^\dagger_{\mathsf{A}}$. It calls its own "oracle" $\mathsf{start\text{-}run}(\mathsf{sid})$.
– When $\mathcal{E}$ calls $\mathsf{start\text{-}challenge}(\mathsf{sid}^*, m_0, m_1)$, $\mathcal{E}'$, if $b^\dagger = 0$, it sends the challenge $(\mathsf{sid}^*, m_0, 0)$ to its challenger. Otherwise, it sends the challenge $(\mathsf{sid}^*, 0, m_1)$. It stores the resulting challenge ciphertext c^* and calls its "oracle" $\mathsf{start\text{-}run}(\mathsf{sid})$.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}next}_{\mathsf{A}}(\mathsf{sid}, M)$, $\mathcal{E}'$ calls its own "oracle" $\mathsf{get\text{-}next}_{\mathsf{A}}(\mathsf{sid}, M)$. If this call does not correspond to the last message of the relevant run of the message-transfer protocol, $\mathcal{E}'$ simply passes the response to $\mathcal{E}$. If this is the last message and $\mathsf{sid} = \mathsf{sid}^*$, $\mathcal{E}'$ sends c^* to $\mathcal{E}$. Otherwise, $\mathcal{E}'$ sets $sk \leftarrow \mathsf{get\text{-}output}_{\mathsf{A}}(\mathsf{sid})$, finds the unique m such that $(\mathsf{sid}, m) \in S^\dagger_{\mathsf{A}}$, and sends $\mathsf{Enc}_{sk}(\mathsf{sid}, m)$ to $\mathcal{E}$.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}next}_{\mathsf{B}}(\mathsf{sid}, M)$, if this is not the last message of the protocol, $\mathcal{E}'$ calls its own "oracle" $\mathsf{get\text{-}next}_{\mathsf{B}}(\mathsf{sid}, M)$ and passes the response to $\mathcal{E}$. If this is the last message, $\mathcal{E}'$ does nothing.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}secrets}$, $\mathcal{E}'$ responds with $(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}})$.
– When $\mathcal{E}$ calls $\mathsf{finalize}(b^*)$, $\mathcal{E}'$ returns b^*.

Let b be the challenge bit of the CPA-security game. If $b = b^\dagger$, then the challenge ciphertext is an encryption of m_b, as in **Game** 2 and $\mathcal{E}'$ correctly outputs b if and only if $\mathcal{E}$ "wins" and $b^* = b^\dagger$. Otherwise, the challenge ciphertext is an encryption of 0 as in **Game** 3 and $\mathcal{E}'$ correctly outputs b if and only if $\mathcal{E}$ "loses." The result follows. (2.2) ■

The result follows.

□

C A suitable malleable commitment scheme

We briefly describe a simple commitment scheme that is statistically hiding, computationally binding, and malleable in the way that we need. It is a basic variant of the Pedersen commitment [Ped92]. We first provide the relevant definitions. For our application, we require that the committed plaintext is a group element and that the commitment can be mauled in such a way that a commitment to group element B can be converted into a commitment to group element B^α. As such, we define a malleable commitment scheme in this specific setting.

Definition 17 (Malleable and rerandomizable commitment). *A* group commitment scheme *is a triple of efficient algorithms,* $(\mathsf{KeyGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$*.* KeyGen *takes as input a description of an abelian group G and outputs public parameters.* Com *takes as the public parameters, a group element $B \in G$, and randomness r, and outputs a commitment C from some commitment space $\mathcal{C}$.* Open *takes as input the public parameters, a* *commitment C and randomness r and outputs an opening x.* Ver *takes as input a commitment C and opening x and outputs either a group element $B \in G$ or the special symbol $\perp$. We often omit explicit reference to the public parameters and/or the randomness. (In the main body, we omit explicit reference to the* Ver *function as well.)*

The scheme is correct *if a commitment opens to the committed message, i.e.,* $\mathsf{Ver}(C, x) = B$ *whenever* $x = \mathsf{Open}(\mathsf{Com}(B))$ *for all $B \in G$. The scheme is* statistically hiding *if for any group element B, the* $\mathsf{Com}(B)$ *is distributed uniformly randomly over the commitment space. The scheme is* computationally binding *if no efficient adversary can produce a commitment C and two openings x, x' such that the two openings verify to different group elements (and not $\perp$). The scheme is* tight *if for each commitment C and group element $B \in G$, there is a unique opening x such that* $\mathsf{Ver}(C, x) = B$.

Such a commitment scheme is rerandomizable *if there exists a pair of efficient algorithms* $(\mathsf{Rerand}, \mathsf{OpenRerand})$ *such that*

1. *for any commitment C and uniformly random r,* $\mathsf{Rerand}(C, r)$ *is uniformly random over the commitment space; and*
2. *for any commitment C and opening x, if* $\mathsf{Ver}(C, x) = B$ *then* $\mathsf{Ver}(C', x') = B$*, where* $C' = \mathsf{Rerand}(C, r)$ *and* $x' = \mathsf{OpenRerand}(x', r)$.

Similarly, such a commitment scheme is malleable *if there exists a pair of efficient algorithms* $(\mathsf{Maul}, \mathsf{OpenMaul})$ *such that for any commitment C and opening x, if* $\mathsf{Ver}(C, x) = B$ *then* $\mathsf{Ver}(C', x') = B$*, where* $C' = \mathsf{Maul}(C, \alpha)$ *and* $x' = \mathsf{OpenMaul}(x', \alpha)$*. In the main body, we omit explicit reference to the functions* $\mathsf{OpenRerand}$ *and* $\mathsf{OpenMaul}$.

We now describe a commitment scheme that suffices for our purposes.

- KeyGen takes as input a group G of order p and returns two uniformly random non-identity group elements g, h.
- Com takes as input a group element $B \in G$ and randomness $(r, s) \in \mathbb{Z}_p^2$ and returns $C := (g^r h^s, h^s B)$.
- Open simply outputs the randomness (r, s).
- Ver takes as input $C = (c_1, c_2)$ and (r, s). It first checks that $c_1 = g^r h^s$. If so, it returns $h^{-s} c_2$.
- Rerand takes as input $C = (c_1, c_2)$ and randomness (r', s') and returns $C' := (g^{r'} h^{s'} c_1, h^{s'} c_2)$.
- $\mathsf{OpenRerand}$ takes as input an opening (r, s) and randomness (r', s') and returns $(r + r', s + s')$.
- Maul takes as input $C = (c_1, c_2)$ and $\alpha \in \mathbb{Z}_p$ and returns $C' := (c_1^\alpha, c_2^\alpha)$.
- $\mathsf{OpenMaul}$ takes as input (r, s) and α and returns $(\alpha r, \alpha s)$.

The following proposition is immediate from inspection and the security of the standard Pedersen commitment.

Proposition 1. *The above commitment scheme is correct, statistically hiding, tight, rerandomizable, and malleable. If the discrete log is hard over G, it is also computationally binding.*

D Proof of Theorem 4

Proof. Item 1 (the security of the underlying protocol without reverse firewalls) follows by a folklore composition theorem. We assume that the last message of the key-agreement protocol is sent by Bob. (This is without loss of generality, as we can always add an empty message from Bob at the end of the protocol.)

Let $\widetilde{A}$ be some functionality-maintaining tampered implementation of Alice in the protocol from Figure 7. Let $\widetilde{\mathsf{A}}^*$ be the "truncation" of $\widetilde{\mathsf{A}}$ to the key-agreement protocol. Note that $\widetilde{\mathsf{A}}^*$ produces valid transcripts (though it may not preserve functionality). Let q be some polynomial bound on the number of oracle calls made by the adversary in the CCA-security game. We assume without loss of generality that the adversary makes no "trivial" calls whose output can be trivially predicted from its previous oracle calls. E.g., it makes no calls to $\mathsf{get\text{-}output_B}(\mathsf{sid})$ where sid does not correspond to a completed run of the protocol, no $\mathsf{get\text{-}next_A}(\mathsf{sid}, \cdot)$ without first calling $\mathsf{start\text{-}run}(\mathsf{sid}, m)$ or $\mathsf{start\text{-}challenge}(\mathsf{sid}, m_0, m_1)$, etc. Consider the following sequence of games.

- **Game** 1 is the CCA-security game against the message-transmission protocol with Alice replaced by $\mathcal{W}_\mathsf{A}^* \circ \widetilde{\mathsf{A}}$.

– **Game** 2 is **Game** 1 in which Alice never sends the final message in a run of the protocol unless Bob's key sk_{B} is well-defined. (I.e., a call to the $\mathsf{return}_{\mathsf{B}}(\mathsf{sid})$ procedure of the underlying key-agreement protocol for the relevant sid does not return $\perp$.)
– For $i = 1, \ldots, q$, **Game** $i+2$ is **Game** $i+1$ in which the final message of the ith run of the protocol is replaced by an encryption of the relevant plaintext m under a uniformly random key.
– **Game** $q+3$ is **Game** $q+2$ in which the final message of the challenge run is replaced by an encryption of 0 under a uniformly random key.
– **Game** $q+4$ is **Game** $q+3$ in which the final message is removed from each run of the protocol and the oracle $\mathsf{get\text{-}output}_{\mathsf{B}}$ is removed.
– For $i = 1, \ldots, q$, **Game** $q+4+i$ is **Game** $q+3+i$ in which $\mathcal{W}_{\mathsf{A}}^* \circ \widetilde{\mathsf{A}}$ is replaced by $\mathcal{W}_{\mathsf{A}}^* \circ \mathsf{A}$ (the honest implementation of Alice composed with the firewall) in the ith run of the protocol.

Note that no adversary can have any advantage in the last game because none of the responses to any of the adversary's queries depend on the challenge bit.

Claim 4.1. *If the encryption scheme has unique ciphertexts, the key-agreement protocol is authenticated for Bob, and $\mathcal{W}_{\mathsf{A}}$ fails detectably and preserves authenticity, then for any PPT adversary $\mathcal{E}$, $|\mathsf{Adv}^{\textbf{\textit{Game}}\ 1}(\mathcal{E}) - \mathsf{Adv}^{\textbf{\textit{Game}}\ 2}(\mathcal{E})|$ is negligible.*

Proof. We construct an adversary $\mathcal{E}'$ in the authentication game against the key-agreement protocol with Alice replaced by $\mathcal{W}_{\mathsf{A}} \circ \widetilde{\mathsf{A}}^*$ as follows. $\mathcal{E}'$ receives as input the public parameters π and passes them to $\mathcal{E}$. $\mathcal{E}'$ then selects $b^\dagger \xleftarrow{\$} \{0,1\}$ uniformly at random and sets $S_{\mathsf{A}}^\dagger, S_{\mathsf{B}}^\dagger, \mathsf{keys} \leftarrow \emptyset$. $\mathcal{E}'$ then responds to oracle calls as follows.

– When $\mathcal{E}$ calls $\mathsf{start\text{-}run}(\mathsf{sid}, m)$, $\mathcal{E}'$ adds (sid, m) to $S_{\mathsf{A}}^\dagger$ and calls its own oracle $\mathsf{start\text{-}run}(\mathsf{sid}, m)$.
– When $\mathcal{E}$ calls $\mathsf{start\text{-}challenge}(\mathsf{sid}, m_0, m_1)$, $\mathcal{E}'$ adds $(\mathsf{sid}, m_{b^\dagger})$ to $S_{\mathsf{A}}^\dagger$ and calls its own oracle $\mathsf{start\text{-}run}(\mathsf{sid}, m_{b_\dagger})$.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}next}_{\mathsf{A}}(\mathsf{sid}, M)$, $\mathcal{E}'$ calls its own oracle $\mathsf{get\text{-}next}_{\mathsf{A}}(\mathsf{sid}, M)$. If this is not the last message of this run of the protocol, $\mathcal{E}'$ then simply passes the resulting message to $\mathcal{E}$. If it is the last message, it checks if the transcript of the underlying key-agreement protocol is valid for $\mathcal{W}_{\mathsf{A}} \circ \mathsf{A}$ (using the efficient algorithm guaranteed by detectable failure). If it is invalid, $\mathcal{E}'$ responds to $\mathcal{E}$ with the special symbol $\perp$. If it is valid, let m be the unique plaintext and index such that $(\mathsf{sid}, m) \in S_{\mathsf{A}}^\dagger$ and let $sk \leftarrow \mathsf{get\text{-}output}_{\mathsf{B}}(\mathsf{sid})$. If $sk = \perp$, then $\mathcal{E}'$ returns the transcript of this run of the key-agreement protocol (and wins the authentication game). Otherwise, it adds (sid, sk) to keys and responds with $\mathsf{Enc}_{sk}(\mathsf{sid}, m)$.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}next}_{\mathsf{B}}(\mathsf{sid}, M)$, if this is not the last message of the protocol, $\mathcal{E}'$ calls its own oracle $\mathsf{get\text{-}next}_{\mathsf{B}}(\mathsf{sid}, M)$ and passes the resulting message to $\mathcal{E}$. If this is the last message of the protocol, it adds (sid, M) to $S_{\mathsf{B}}^\dagger$ and sends nothing to $\mathcal{E}$.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}output}_{\mathsf{B}}(\mathsf{sid})$, $\mathcal{E}'$ finds the unique M and sk such that $(\mathsf{sid}, M) \in S_{\mathsf{B}}$ and $(\mathsf{sid}, sk) \in \mathsf{keys}$. It computes $\mathsf{Dec}_{sk}(\mathsf{sid}, M)$ and responds with the result.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}secrets}$, $\mathcal{E}'$ calls its own oracle $\mathsf{get\text{-}secrets}$ and passes the result to $\mathcal{E}$.
– When $\mathcal{E}$ calls $\mathsf{finalize}(b^*)$, $\mathcal{E}'$ simply terminates.

Note that the view of $\mathcal{E}$ is identical to its view in both **Game** 1 and **Game** 2 unless at some point it constructs a valid transcript such that sk_{B} is not well-defined. If it does construct such a transcript, then $\mathcal{E}'$ wins the authentication game. The result follows. (4.1) ■

Claim 4.2. *If the key-agreement protocol is secure against active adversaries for Alice, the encryption scheme has unique ciphertexts, and $\mathcal{W}_{\mathsf{A}}$ preserves Alice's security and fails detectably, then for any PPT adversary $\mathcal{E}$, $|\mathsf{Adv}^{\textbf{\textit{Game}}\ i+1}(\mathcal{E}) - \mathsf{Adv}^{\textbf{\textit{Game}}\ i+2}(\mathcal{E})|$ is negligible.*

Proof. We construct an adversary $\mathcal{E}'$ in the security game against the key-agreement protocol with Alice replaced by $\mathcal{W}_{\mathsf{A}} \circ \widetilde{\mathsf{A}}^*$ as follows. $\mathcal{E}'$ receives as input the public parameters π and passes them to $\mathcal{E}$. $\mathcal{E}'$ then selects $b^\dagger \xleftarrow{\$} \{0,1\}$ uniformly at random and sets $S_{\mathsf{A}}^\dagger, S_{\mathsf{B}}^\dagger, \mathsf{keys} \leftarrow \emptyset$ and $j \leftarrow 0$. It responds to the oracle calls of $\mathcal{E}$ as follows.

– When $\mathcal{E}$ calls $\mathsf{start\text{-}run}(\mathsf{sid}, m)$, $\mathcal{E}'$ adds (sid, j, m) to $S_{\mathsf{A}}^\dagger$. If $j = i$, it calls its own oracle $\mathsf{start\text{-}challenge}(\mathsf{sid}^*, m)$; otherwise it calls $\mathsf{start\text{-}run}(\mathsf{sid}, m)$. Finally, it increments j.

– When $\mathcal{E}$ calls $\mathsf{start\text{-}challenge}(\mathsf{sid}, m_0, m_1)$, $\mathcal{E}'$ adds $(\mathsf{sid}, m_{b^\dagger})$ to $S^\dagger_{\mathsf{A}}$ and (sid, j) to $S^\dagger_{\mathsf{B}}$. If $j = i$, it calls its own oracle $\mathsf{start\text{-}challenge}(\mathsf{sid}^*, m_{b_\dagger})$; otherwise it calls $\mathsf{start\text{-}run}(\mathsf{sid}, m_{b_\dagger})$. Finally, it increments j.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}next_A}(\mathsf{sid}, M)$, $\mathcal{E}'$ calls its own oracle $\mathsf{get\text{-}next_A}(\mathsf{sid}, M)$. if this is not the last message of this run of the protocol, $\mathcal{E}'$ then simply passes the resulting message to $\mathcal{E}$. If it is the last message, it checks if the transcript of the underlying key-agreement protocol is valid for $\mathcal{W}_{\mathsf{A}} \circ \mathsf{A}$ (using the efficient algorithm guaranteed by detectable failure). If it is invalid, $\mathcal{E}'$ responds to $\mathcal{E}$ with the special symbol $\perp$. If it is valid, let m, k be the unique plaintext and index such that $(\mathsf{sid}, k, m) \in S^\dagger_{\mathsf{A}}$. If $k < i$, $\mathcal{E}'$ selects $sk \xleftarrow{\$} \mathcal{K}$. If $k \geq i$, $\mathcal{E}'$ sets $sk \leftarrow \mathsf{get\text{-}output_B}(\mathsf{sid})$. Finally, it responds to $\mathcal{E}$ with the message $\mathsf{Enc}_{sk}(\mathsf{sid}, m)$.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}next_B}(\mathsf{sid}, M)$, $\mathcal{E}'$ calls its own oracle $\mathsf{get\text{-}next_B}(\mathsf{sid}, M)$ and passes the resulting message to $\mathcal{E}$. If this is the last message of the protocol, it also adds (sid, M) to $S^\dagger_{\mathsf{B}}$.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}output_B}(\mathsf{sid})$, $\mathcal{E}'$ finds the unique M and sk such that $(\mathsf{sid}, M) \in S^\dagger_{\mathsf{B}}$ and $(\mathsf{sid}, sk) \in \mathsf{keys}$. It computes $\mathsf{Dec}_{sk}(\mathsf{sid}, M)$ and responds with the result.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}secrets}$, $\mathcal{E}'$ calls its own oracle $\mathsf{get\text{-}secrets}$ and passes the result to $\mathcal{E}$.
– When $\mathcal{E}$ calls $\mathsf{finalize}(b^*)$, $\mathcal{E}'$ returns 1 if $b^\dagger = b^*$ and 0 otherwise.

Suppose the challenge bit b in the key-agreement security game is 0 so that the challenge key sk^* corresponding to the ith run of the protocol was selected uniformly at random or is the special symbol $\perp$. Then, clearly the view of $\mathcal{E}$ is identical to its view in **Game** $i+2$, and $\mathcal{E}'$ correctly outputs 0 if and only if $\mathcal{E}$ "loses" its simulated game by returning $b^\dagger = b^*$. If, on the other hand, the challenge bit b is 1, then the view of $\mathcal{E}$ is identical to its view in **Game** $i+1$, and $\mathcal{E}'$ correctly outputs 1 if and only if $\mathcal{E}$ "wins" its simulated game. The result follows.

(4.2) ■

Claim 4.3. *If the encryption scheme is CCA-secure, then for any PPT adversary $\mathcal{E}$, $|\mathsf{Adv}^{\textbf{\textit{Game}}\ q+2}(\mathcal{E}) - \mathsf{Adv}^{\textbf{\textit{Game}}\ q+3}(\mathcal{E})|$ is negligible.*

Proof. We construct an adversary $\mathcal{E}'$ in the CCA-security game against the encryption scheme as follows. $\mathcal{E}'$ first runs the setup procedure of the key-agreement protocol, receiving as output σ_{A}, σ_{B}, and π. $\mathcal{E}'$ will simulate many runs of the key-agreement protocol with Alice replaced by $\mathcal{W}_{\mathsf{A}} \circ \widetilde{\mathsf{A}}^*$ and input $(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}}, \pi)$. For convenience, we give $\mathcal{E}'$ an "oracle interface" to these simulated runs with "oracle" calls $\mathsf{start\text{-}run}$, $\mathsf{get\text{-}next_A}$, and $\mathsf{get\text{-}next_B}$ as in the key-agreement security game (Figure 3). $\mathcal{E}'$ sends π to $\mathcal{E}$, selects $b^\dagger \xleftarrow{\$} \{0,1\}$ uniformly at random, and sets $S_{\mathsf{A}}, S_{\mathsf{B}}, \mathsf{keys} \leftarrow \emptyset$. It then responds to the oracle queries of $\mathcal{E}$ as follows.

– When $\mathcal{E}$ calls $\mathsf{start\text{-}run}(\mathsf{sid}, m)$, $\mathcal{E}'$ adds (sid, m) to S_{A}. It calls its own "oracle" $\mathsf{start\text{-}run}(\mathsf{sid}, m)$.
– When $\mathcal{E}$ calls $\mathsf{start\text{-}challenge}(\mathsf{sid}^*, m_0, m_1)$, if $b^\dagger = 0$, it $\mathcal{E}'$ sends the challenge $(\mathsf{sid}^*, m_0, 0)$ to its challenger. Otherwise, it sends the challenge $(\mathsf{sid}^*, 0, m_1)$. It stores the resulting challenge ciphertext c^*. and calls its "oracle" $\mathsf{start\text{-}run}(\mathsf{sid}, m_{b^\dagger})$.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}next_A}(\mathsf{sid}, M)$, if this call does not correspond to the last message of the relevant run of the message-transmission protocol, $\mathcal{E}'$ calls its own "oracle" $\mathsf{get\text{-}next_A}(\mathsf{sid}, M)$ and passes the response to $\mathcal{E}$. If the transcript of the underlying key-agreement protocol is invalid, then $\mathcal{E}'$ sends $\perp$ to $\mathcal{E}$. If it is valid and $\mathsf{sid} = \mathsf{sid}^*$, it sends c^* to $\mathcal{E}$. Otherwise, $\mathcal{E}'$ selects a key $sk \xleftarrow{\$} \mathcal{K}$ uniformly at random, adds (sid, sk) to keys, finds the unique m such that $(\mathsf{sid}, m) \in S_{\mathsf{A}}$, and sends $\mathsf{Enc}_{sk}(\mathsf{sid}, m)$ to $\mathcal{E}$.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}next_B}(\mathsf{sid}, M)$, if this is not the last message of the protocol $\mathcal{E}'$ calls its own "oracle" $\mathsf{get\text{-}next_B}(\mathsf{sid}, M)$ and passes the response to $\mathcal{E}$. If this is the last message of the protocol, it adds (sid, M) to S_{B}.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}output_B}(\mathsf{sid})$, $\mathcal{E}'$ finds the unique M and sk such that $(\mathsf{sid}, M) \in S_{\mathsf{B}}$ and $(\mathsf{sid}, sk) \in \mathsf{keys}$. It computes $\mathsf{Dec}_{sk}(\mathsf{sid}, M)$ and responds with the result.
– When $\mathcal{E}$ calls $\mathsf{get\text{-}secrets}$, $\mathcal{E}'$ responds with $(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}})$.
– When $\mathcal{E}$ calls $\mathsf{finalize}(b^*)$, $\mathcal{E}'$ returns b^*.

Let b be the challenge bit in the CPA-security game against $\mathcal{E}'$. If $b^\dagger = b$, the view of $\mathcal{E}$ is identical to its view in **Game** 2. In this case, $\mathcal{E}'$ wins if and only if $\mathcal{E}$ wins the simulated **Game** 2. If $b^\dagger \neq b$, the view of $\mathcal{E}$ is identical to its view in **Game** 3 and $\mathcal{E}'$ wins if and only if $\mathcal{E}$ loses the simulated game. The result follows. (4.3) ■

It should be clear that any adversary in game **Game** $q+3$ can be easily converted into an adversary in **Game** $q+4$ with same advantage.

Claim 4.4. *If $\mathcal{W}_{\mathsf{A}}$ is exfiltration resistant against* B *with valid transcripts then for any PPT adversary $\mathcal{E}$, $|\mathsf{Adv}^{\textbf{\textit{Game}}\ q+3+i}(\mathcal{E}) - \mathsf{Adv}^{\textbf{\textit{Game}}\ q+4+i}(\mathcal{E})|$ is negligible.*

Proof. Let $\widetilde{\mathsf{A}}^{(m)}$ be $\widetilde{\mathsf{A}}^*$ with input plaintext fixed to m.

We construct an adversary $\mathcal{E}'$ in LEAK (Figure 1) as follows. $\mathcal{E}'$ first runs the setup procedure of the key-agreement protocol, receiving as output σ_{A}, σ_{B}, and π. As above, $\mathcal{E}'$ will simulate many runs of the key-agreement protocol with Alice replaced by $\mathcal{W}_{\mathsf{A}} \circ \mathsf{A}$ and input $(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}}, \pi)$. So, for convenience, we give $\mathcal{E}'$ an "oracle interface" to these simulated runs with "oracle" calls $\mathsf{start\text{-}run}$, $\mathsf{get\text{-}next}_{\mathsf{A}}$ and $\mathsf{get\text{-}next}_{\mathsf{B}}$ as in the key-agreement security game (Figure 3). $\mathcal{E}'$ selects $b^\dagger \xleftarrow{\$} \{0,1\}$ uniformly at random, sets $j \leftarrow 1$ and $\mathsf{ids} \leftarrow \emptyset$, and simulates a run of $\mathcal{E}$, responding to oracle calls as follows.

- When $\mathcal{E}$ calls $\mathsf{start\text{-}run}(\mathsf{sid}, m)$, $\mathcal{E}'$ adds (sid, j) to ids. If $j \neq i$, it calls its own "oracle" $\mathsf{start\text{-}run}(\mathsf{sid}, m)$ and increments j. If $j = i$, $\mathcal{E}'$ increments j and constructs the circuit $\widetilde{B}$ described below with sid^*, $(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}}, \pi)$, $b^\dagger$, j, ids, the state of $\mathcal{E}$, and the state of the various "oracles" hard-coded into it. It then returns $(\widetilde{\mathsf{A}}^{(m_{b^\dagger})}, \widetilde{B}, (\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}}, \pi))$.
- When $\mathcal{E}$ calls $\mathsf{get\text{-}next}_{\mathsf{A}}(\mathsf{sid}, M)$, $\mathcal{E}'$ calls its own "oracle" $\mathsf{get\text{-}next}_{\mathsf{A}}(\mathsf{sid}, M)$ and passes the response to $\mathcal{E}$.
- When $\mathcal{E}$ calls $\mathsf{get\text{-}next}_{\mathsf{B}}(\mathsf{sid}, M)$, $\mathcal{E}'$ calls its own "oracle" $\mathsf{get\text{-}next}_{\mathsf{B}}(\mathsf{sid}, M)$ and passes the response to $\mathcal{E}$.
- When $\mathcal{E}$ calls $\mathsf{start\text{-}challenge}(\mathsf{sid}^*, m_0, m_1)$,
- When $\mathcal{E}$ calls $\mathsf{get\text{-}secrets}$, $\mathcal{E}'$ responds with $(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}})$.

$\widetilde{B}$ will play the role of Bob in the key-agreement protocol, and it has the state of $\mathcal{E}$ and the "oracles" hard-coded into it. It can make "oracle" calls to simulated protocols with Alice replaced by $\mathcal{W}_{\mathsf{A}} \circ \mathsf{A}$. It also starts its own "oracle" simulations with Alice replaced by $\mathcal{W}_{\mathsf{A}} \circ \widetilde{\mathsf{A}}^*$ instead. To distinguish these oracles, we write, e.g., $\mathsf{get\text{-}next}_{\mathcal{W}_{\mathsf{A}} \circ \widetilde{\mathsf{A}}^*}$ and $\mathsf{get\text{-}next}_{\mathcal{W}_{\mathsf{A}} \circ \mathsf{A}}$. Note that $\widetilde{B}$ is itself playing a game in which it exchanges its own messages with the challenge party A^* in LEAK (Figure 1). $\widetilde{B}$ continues to simulate $\mathcal{E}$ from its current state, responding to oracle calls as follows.

- When $\mathcal{E}$ calls $\mathsf{start\text{-}run}(\mathsf{sid}, m)$, $\widetilde{\mathsf{B}}$ adds (sid, j) to ids, increments j, and calls its own "oracle" $\mathsf{start\text{-}run}(\mathsf{sid}, m)$.
- When $\mathcal{E}$ calls $\mathsf{get\text{-}next}_{\mathsf{A}}(\mathsf{sid}, M)$, $\widetilde{B}$ finds the unique k such that $(\mathsf{sid}, k) \in \mathsf{ids}$. If $k < i$, $\widetilde{\mathsf{B}}$ calls its "oracle" $\mathsf{get\text{-}next}_{\mathcal{W}_{\mathsf{A}} \circ \mathsf{A}}(\mathsf{sid}, M)$ and passes the response to $\mathcal{E}$. If $k = i$, then $\widetilde{\mathsf{B}}$ sends the message M to the challenge party A^* and passes the response to $\mathcal{E}$. If $k > i$, then $\widetilde{\mathsf{B}}$ calls its "oracle" $\mathsf{get\text{-}next}_{\mathcal{W}_{\mathsf{A}} \circ \widetilde{\mathsf{A}}^*}(\mathsf{sid}, M)$ and passes the response to $\mathcal{E}$.
- When $\mathcal{E}$ calls $\mathsf{get\text{-}next}_{\mathsf{B}}(\mathsf{sid}, M)$, $\widetilde{\mathsf{B}}$ calls its own "oracle" $\mathsf{get\text{-}next}_{\mathsf{B}}(\mathsf{sid}, M)$ and passes the response to $\mathcal{E}$.
- When $\mathcal{E}$ calls $\mathsf{finalize}(b^*)$, $\widetilde{\mathsf{B}}$ sets its state to 0 if $b^* = b^\dagger$ and to 1 otherwise.
- When $\mathcal{E}$ calls $\mathsf{get\text{-}secrets}$, $\widetilde{\mathsf{B}}$ responds with $(\sigma_{\mathsf{A}}, \sigma_{\mathsf{B}})$.

Finally, $\mathcal{E}'$ receives the state of $\widetilde{B}$ and simply returns its value.

Let b be the challenge bit in LEAK. Then, if $b = 0$ so that the challenge party is honest, the view of $\mathcal{E}$ is identical to its view in **Game** $q+5$. Then, the final state of $\widetilde{\mathsf{B}}$ matches b if and only if $\mathcal{E}$ wins this simulated game. If, on the other hand, $b = 1$, then the view of $\mathcal{E}$ is identical to its view in **Game** $q+4$, and the final state of $\widetilde{\mathsf{B}}$ matches b if and only if $\mathcal{E}$ loses this simulated game. The result follows. (4.4) ■

□

E Proof of Theorem 9

Proof of Theorem 9. Let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure encryption scheme with some reverse firewall $\mathcal{W}$. Note that we can view $\mathcal{W}$ as a map between ciphertexts.

We present a one-bit PKE scheme as follows. Let m_0, m_1 be distinct plaintexts. The public key is then $(e_0 = \mathcal{W}(\mathsf{Enc}_{sk}(m_0)), e_1 = \mathcal{W}(\mathsf{Enc}_{sk}(m_1)))$, and the secret key is just the secret key of the underlying scheme. To encrypt a bit b, we compute $\mathcal{W}(e_b)$. The decryption algorithm of the public-key scheme runs the decryption algorithm of the symmetric-key scheme Dec and outputs 0 if the result is m_0, 1 if it is m_1, and $\perp$ otherwise.

Let $\mathcal{E}$ be a PPT adversary in the semantic-security game against the above scheme. We assume without loss of generality that $\mathcal{E}$ never outputs a pair of identical challenge plaintexts. We construct an efficient tampered encryption algorithm $\bar{\mathsf{Enc}}$ and an efficient adversary $\mathcal{E}'$ in the CPA-security game against $\mathcal{W} \circ \bar{\mathsf{Enc}}$. Choose $m_0^\dagger$ and $m_1^\dagger$ uniformly at random from the plaintext space. Simulate $\mathcal{E}$ polynomially many times and let i such in at least polynomial many of these runs, the challenge plaintexts chosen by $\mathcal{E}$ differ in the ith bit. Fix $c_0 = \mathsf{Enc}_{sk}(m_0)$ and $c_1 = \mathsf{Enc}_{sk}(m_1)$. Then, we define $\bar{\mathsf{Enc}}_{sk}(m_b^\dagger) = c_b$ and for all other plaintexts m, $\bar{\mathsf{Enc}}(m) = \mathcal{W}(c_b)$ where b is the ith bit of m. Then, $\mathcal{E}'$ behaves as follows.

- It calls the encryption oracle on input $m_0^\dagger$ and $m_1^\dagger$. Call the results e_0 and e_1.
- It runs $\mathcal{E}$ with input $pk = (e_0, e_1)$, receiving as output two challenge plaintexts, (m_0^*, m_1^*). $\mathcal{E}'$ then outputs these as its own challenge plaintexts.
- On input c^*, a challenge ciphertext, $\mathcal{E}'$ passes c^* to $\mathcal{E}$, receiving as output a bit b.
- If m_0^* and m_1^* differ in their ith bit and are distinct from $m_0^\dagger$ and $m_1^\dagger$, output the bit corresponding to the plaintext whose ith bit is b. Otherwise, return a uniformly random bit.

Note that the view of $\mathcal{E}$ is identical to its view in the semantic security game against the public-key scheme. Furthermore, with non-negligible probability, we have that $m_0^\dagger$, $m_1^\dagger$, m_0^*, and m_1^* are distinct and m_0^* and m_1^* differ in their ith bit. When both of these conditions are satisfied, $\mathcal{E}'$ guesses correctly if and only if $\mathcal{E}$ guesses correctly. The result follows. $\square$