On The Use of Remote Attestation to Break and Repair Deniability

Lachlan J. Gunn
Aalto University
lachlan.gunn@aalto.fi

Ricardo Vieitez Parra
Aalto University
ricardo.vieitezparra@aalto.fi

N. Asokan
Aalto University
asokan@acm.org

ABSTRACT

Deniable messaging protocols allow two parties to have 'off-the-record' conversations without leaving any record that can convince external verifiers about what either of them said during the conversation. Recent events like WikiLeaks email dumps underscore the importance of deniable messaging to whistleblowers, politicians, dissidents and many others. Consequently, messaging protocols like Signal and OTR are expressly designed to provide deniability.

Many commodity devices today support hardware-assisted *remote attestation* which can be used to convince a remote verifier of some property locally observed on the device.

We show how an adversary can use remote attestation to *undetectably* break deniability in any deniable protocol (including messaging protocols) that provide an authenticated channel. We prove that our attack allows an adversary to convince *skeptical verifiers* and describe a concrete implementation of the attack against the Signal messaging protocol. We then show how attestation itself can be used to restore deniability by thwarting a realistic class of adversaries from mounting such attacks.

Hardware-based attestation changes the adversary model for deniable protocols, and its availability has now made it entirely practical for well-resourced attackers to break deniability, completely unbeknownst to the victim.

1 INTRODUCTION

There is a growing trend towards the use of communications dumps as political weapons. Transparent insertion of signatures by mail servers as an anti-spam measure [21] has made email dumps into potent weapons, as they allow readers to verify the authenticity of emails leaked by unknown or untrusted parties [3].

A *deniable* [20] but authenticated communications channel allows the sender of a message to authenticate themselves to the recipient without the possibility for anyone else to reliably authenticate the source of the message, even with the aid of the original intended recipient. Modern secure messaging protocols [10, 25] place great emphasis on supporting deniability. These have become popular in the wake of the Snowden disclosures [13], and in particular amongst politicians following a number of well-known email dumps [17]. Thus it is reasonable to expect that when someone wants to have a conversation without leaving a verifiable audit trail—such as when a whistleblower talks to a journalist—they may choose to use a modern deniable messaging protocol like Signal, rather than a medium such as email.

Hardware-based *Trusted Execution Environments (TEEs)* like ARM TrustZone and Intel SGX are widely available in commodity devices. They can support *remote attestation*: the ability to convince a remote verifier about properties observable locally on the device.

Deniability depends upon the ability of an adversary to lie. Remote attestation allows even adversaries to prove that they behaved correctly. This changes the adversary model: in cryptographic protocols, an honest adversary is not a threat. Remote attestation renders honest adversaries a realistic threat for deniable protocols. In this paper, we show that an adversary can use remote attestation on a device (say Bob's) to produce a publicly verifiable, non-repudiable transcript of an otherwise deniable protocol run thus breaking deniability for the communication peer (Alice). Worse still, Alice *cannot detect* her loss of deniability. We provide a formal proof that the transcript resulting from the attack can convince a *skeptical verifier* (e.g., a journalist who does not trust Bob or some recording software on Bob's device) of what Alice[1] said during the conversation. Furthermore, the transcript is *transferable*: the verifier is not requried to be online during the communication session. This is at odds with the expectations of users of deniable messaging protocols, who assume that a remote adversary cannot obtain verifiable transcripts of their messages by compromising their contacts' devices.

We have implemented a working prototype of the attack against the Signal messaging protocol using Intel SGX for remote attestation. But the basic approach can be used to attack any deniable protocol that makes use of an authenticated channel. We discuss several such examples.

We show that remote attestation itself can be used to restore deniability for Alice by thwarting a realistic class of adversaries which we call *software-modifying adversary* (who can install or manipulate software on Bob's device but cannot install new TEEs) from mounting this attack. The intuition behind the defense is for Bob's device to attest to Alice either that it will make no further attestations about the conversation, or that the message authentication key(s) used in the session are present *outside* the TEE thus any subsequent attested transcript from Bob's device will not convince skeptical verifiers about the origin of the messages Alice sends in the session. We show that the attack cannot be defended against stronger adversaries (who can install TEEs on Bob's device) without foregoing sender authentication in the messaging protocol.

Finally, we show that the central idea of attesting confidentiality (or lack thereof) and behavior of secret keys can have positive applications, too. One such application is to use a TEE to 'upgrade' a shared-key based message authentication code to a publicly verifiable signature which may be useful in scenarios where resource-constrained devices (e.g., automotive microcontroller units) need to produce publicly verifiable statements (e.g., for use in accident investigation).

Our contributions are as follows:

[1] As identified by the long-term identity key that she uses to authenticate herself to peers in the deniable messaging protocol.

- **Breaking deniability:** We present a generic method for breaking deniability in messaging protocols with sender authentication [Section 3.2] and a concrete implementation of it using Intel SGX, targeting the Signal messaging protocol [Section 3.3]. We discuss several other types of deniable protocols which can be similarly attacked [Section 3.4].
- **Restoring deniability:** We show how we can restore deniability (a) in the presence of adversaries who can only modify software, by using remote attestation itself [Section 4.1], and (b) in the presence of stronger adversaries, by foregoing sender authentication [Section 4.2].
- **Impossibility of deniability without attestability:** We formally prove (a) that the attack results in a transcript that can convince skeptical, offline verifiers [Theorem 5.2], (b) that *any* authenticated messaging protocol that does not use a TEE can be undetectably rendered non-repudiable [Theorem 5.3], and (c) that it is not possible to defend against a hardware-modifying adversary without sacrificing sender authentication in the messaging protocol [Corollary 5.4].
- **Positive uses:** We show that the basic pattern used in the attack has positive applications such as allowing a TEE to 'upgrade' a shared-key based message authenticator to a publicly verifiable signature [Section 6].

We emphasize that we do not claim to have broken existing deniable messaging protocols. Rather, we want to highlight how the assumptions behind the design of such protocols have changed with the widespread availability of remote attestation in commodity devices. We hope that our work will help protocol designers to be cognizant of how this change affects the guarantees that their protocols provide in the real world.

Though we focus on deniable messaging, this observation applies to the more general zero-knowledge setting; the use of remote attestation effectively turns interactive protocols into non-interactive ones, allowing the verifier in a zero-knowledge protocol to prove to a third party any property that it can locally verify.

2 PRELIMINARIES

2.1 Deniable protocols

We consider the following setting for secure messaging protocols: two parties, Alice and Bob, each having long-term identity keys. The messaging scheme provides the usual authenticity, integrity, and confidentiality guarantees to Alice and Bob [32]. Suppose that one party (say Bob) has recording software (which we refer to as the *prover*) installed on his device (possibly by an external adversary without Bob's knowledge). The goal of the adversary is to use a protocol transcript recorded on Bob's device to convince a *skeptical* third party *verifier* (Valerie) that a certain message was definitely sent by Alice, the *victim*. Valerie is "skeptical" in the sense that she does not automatically believe the claims of provers since provers may be dishonest. Valerie therefore expect that the claims are backed up by verifiable evidence in the transcripts.

Informally, a *deniable* protocol prevents the prover from obtaining such evidence. This is not necessarily at odds with the requirement for authentication; a protocol can provide strong authentication between its participants, while at the same time not allowing either party to prove anything about it to anyone else.

Deniability is traditionally established by showing that an adversary can produce a protocol transcript, indistinguishable from a real one, consisting of arbitrary messages of the adversary's choice. This is known as *off-line deniability* [16]. The double-ratchet algorithm [24] used by Signal has this property: anyone can construct a completely valid transcript for any set of messages between any two parties. Other protocols such as TLS [15] are also deniable; in both cases, message authentication uses symmetric-key cryptography; thus each party can produce a transcript containing arbitrary messages purporting to be from the other, along with correct message authentication tokens.

A stronger notion than off-line deniability is *on-line* deniability [16, 32, 33] where the prover is allowed to communicate with the verifier *during* the protocol. In general, this is much harder to achieve, though protocols such as those by [16, 33] have had some success. Note that the proof obtained by the verifier performing an online attack in this case is *not transferable*, meaning that it cannot, for example, be published in a data dump to implicate the victim in the eyes of skeptical observers.

2.2 Hardware-assisted trusted execution environments

A TEE is a security primitive that makes it possible to execute security-critical logic isolated from all other software on the same device. In addition, TEEs support secure persistent storage, referred to as *sealed storage*, for persistently storing sensitive data like keys, and *remote attestation*, the possibility of convincing a remote verifier of the configuration or other properties of the device. Over the past two decades, processor extensions to enable TEEs have become widely deployed. ARM TrustZone [1] (common on smartphones and tablets) and Intel SGX [2] (for x86-based personal computers and servers) are two examples. Trusted Platform Modules (TPMs) [18], typically realized as discrete components, are an example of a widely deployed type of *fixed-function* TEE.

Intel SGX allows a developer to designate a (security-critical) portion of an application as an *enclave*. When an enclave is initialized, the processor measures the enclave. Data belonging to the enclave are automatically protected when they leave the processor, ensuring that only the enclave code can access its data. Memory protection provided by SGX ensures that enclaves are strongly isolated even from the operating system.

TPMs provide an append-only log that is used to store 'measurements' of subsequent components of the boot process. A *root of trust for measurement* appends a hash of the BIOS, which appends a hash of the bootloader, and with operating system support this measurement chain can continue as far as user applications. A TPM provides only a chain of measurement, not any form of memory protection, but this is sufficient to perform remote attestation.

Remote attestation is the process by which a TEE on a device takes part in a secure protocol in order to convince a remote verifier about specific properties that can be observed on the device. The most common form of remote attestation is to convince the verifier of the software state of the local device. This is done by measuring the software running locally and signing it with a key known only to the TEE. The manufacturers of a TEE typically issue a certificate for the TEE's signature verification key. A verifier who trusts a TEE manufacturer and knows the manufacturer's signature verification key, can verify the attestation from a TEE from that manufacturer to convince itself of the state of the device. Any locally observable property, such as the result of running a program in a TEE, can be conveyed via remote attestation.

Attestation of an SGX enclave consists of a number of components [2, §2.15], including the code signature verification key that was used verify the enclave, the enclave's measurement hash, and a piece of arbitrary data provided by the attesting enclave from within its protected memory region. The utility of this attestation depends upon the isolation guarantees provided by the processor; a production-mode SGX enclave is strongly isolated from outside code, and its state is therefore mutated according only to the rules of the enclave. This allows us to make more detailed inferences about the state of the enclave.

2.3 Universal composability

We perform our analysis in the *universal composability* framework [11]. This framework defines security according to the inability of the environment, which controls the inputs and the adversary, and observes the outputs of the protocol, to distinguish between a real run of the protocol and one that simply calls the ideal functionality that the real-world protocol attempts to emulate.

If a protocol π is indistinguishable from another ρ in this way, we say that π UC-emulates ρ, and vice-versa. This approach is particularly attractive because of what is known as the *universal composition theorem* [11, Theorem 13], by which we are assured that a protocol composed of many sub-protocols is indistinguishable from a similar protocol in which all of its sub-protocols are replaced by the ideal functionalities that they UC-emulate.

3 BREAKING DENIABILITY WITH REMOTE ATTESTATION

The goal of a deniable messaging protocol is to allow its participants to communicate in such a way that the recipient of a message in a protocol session can be assured of the identity of the sender, but that outsiders cannot be so-assured, even with the cooperation of the original recipient.

Recall that remote attestation makes it possible to transform any *locally verifiable property* on a system into an unforgeable statement that can be—possibly publicly—*verified by a remote party*. In this attack, we select as the attested property the *output of a protocol*, as implemented by a program $\mathcal{P}$. This results in a statement of the form:

Program $\mathcal{P}$, running under conditions [...], output x.

The intuition behind our attack is simple: if the output of $\mathcal{P}$ convinces the party executing it of any statement, then the attestation convinces its verifier of the same statement. If $\mathcal{P}$ implements an authenticated message functionality, then the attestation can convince anyone verifying it that a particular party sent a particular message. In this section, we describe our deniability-breaking protocol transformation, and its concrete realization using Intel SGX, targeting Signal.

3.1 Adversary model for deniability

The traditional adversary model for deniable communication in the presence of skeptical verifiers is as follows. The adversary is assumed *not to be able* to compromise either the victim's (Alice) or the verifier's (Valerie) devices (if the adversary compromises Alice's device, then he can use it to send legitimate messages saying whatever he wants!). The adversary is assumed to have access to the device of the person the victim is communicating with (Bob).

In this paper, we make a distinction between two kinds of adversaries. A remote attacker can install or manipulate software on Bob's device but cannot modify the hardware. We call this a *software-modifying adversary*. Conversely, an attacker with physical access to Bob's device can modify its hardware, and thus nest one execution environment within another. This type of attacker we call a *hardware-modifying adversary*. We assume that the adversary cannot compromise the integrity of TEEs.

This distinction is important if the security of a protocol depends upon the absence of some piece of hardware—an attacker wishing to insert a new piece of hardware into a device must have physical access, whereas a remote attacker must content themselves with whatever happens to be available.

In either case, the adversary can use all TEEs on the devices under its control, and in particular can produce remote attestations that are trusted by the skeptical verifier Valerie. As a pre-requisite for Valerie to be convinced by the remote attestation from Bob's device, we assume that Valerie has securely obtained—and trusts the integrity of—the trust root for verifying the attestation, e.g. the signature verification key from the manufacturer of the TEE. Importantly, the adversary's remote attestations do not need to convince everyone, but only those verifiers that Alice wants to keep from learning what messages she sent.

3.2 A deniability-breaking protocol transformation

We show our protocol transformation in Figure 1. It takes the original protocol π and adds an attestation by a TEE on Bob's device to the original protocol's output after running it. It then sends this output along with the attestation to the verifier, Valerie, who verifies the attestation and then outputs the value sent by Bob's device. We denote the transformed protocol as $\mathsf{Clone}_B(\pi)$. Valerie knows that the message from Bob's TEE is authentic, because the verification function $\mathsf{Verify}_{\mathcal{P}}(x, \sigma)$ ensures that the message x was emitted by the program $\mathcal{P}$ implementing π. These changes are invisible to Alice, who just sees a normal execution of π.

This result is significant in that it allows us to non-interactively prove to a skeptical verifier that a protocol has yielded some result. Where this protocol attempts to realize a deniable communications channel, this is catastrophic to its security.

Canetti [11] defined the secure message transmission functionality $\mathcal{F}_{\text{SMT}}$; this functionality allows a party P to send a message $(\mathsf{Send}, sid, Q, m)$, and it will accordingly send $(\mathsf{Sent}, sid, P, m)$ to party Q. Significantly, a user cannot in general return a message to the functionality for after-the-fact verification of its origin. As a result, the recipient of a message can lie about what they received, making messages sent through the functionality *deniable*. Critically, deniability is a feature of the protocol realizing $\mathcal{F}_{\text{SMT}}$, and not of the

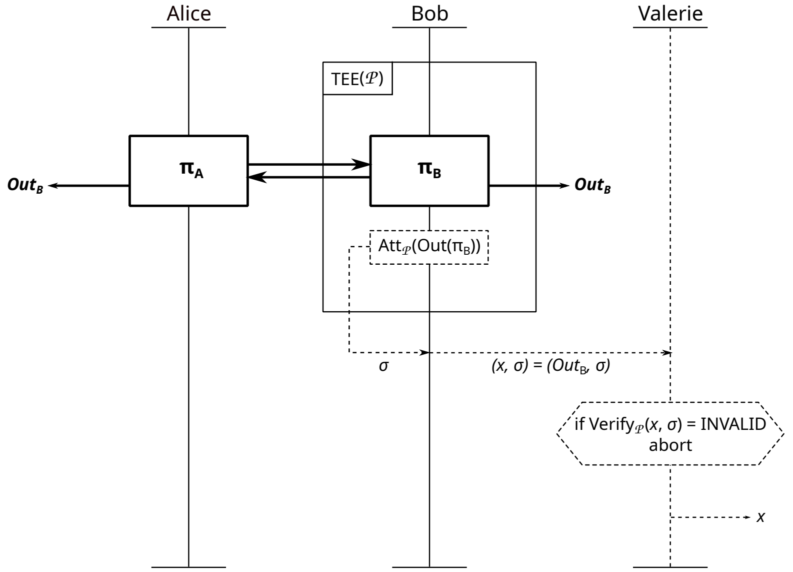
Alice
Bob
Valerie
TEE($\mathcal{P}$)
π_A
π_B
Out_B
Out_B
Att$_{\mathcal{P}}$(Out(π_B))
σ
$(x, \sigma) = (Out_B, \sigma)$
if Verify$_{\mathcal{P}}$(x, σ) = INVALID
abort
x

Figure 1: Our modified protocol $\mathrm{Clone}_B(\pi)$, with a TEE added and additions to the protocol shown in dashed lines. In this three-party protocol, we take the original protocol $\pi = (\pi_A, \pi_B)$—which does not use the TEE shown—and convert Bob's part into a program $\mathcal{P}$ that executes his component π_B before producing an attestation to its output. No change is made to Alice's part of the protocol—the modifications are completely invisible to her. Valerie can then verify the attestation, and knowing that the program $\mathcal{P}$ attests to the output of π_B, concludes that the protocol was correctly executed. The protocol is so-named because, in general, it takes a functionality and 'clones' one of its outputs, so that Valerie will always output the same value as the one being cloned.

functionality; there exist both deniable and non-repudiable protocols that realize this functionality. For example, we might imagine a protocol that has Alice sign every message to Bob, such as that described in [12, §4.1]. Conversely, protocols such as Signal, TLS, and Off-the-Record provide deniable realizations of $\mathcal{F}_{\mathrm{SMT}}$.

This fact means that there is no guarantee that deniability will be preserved under composition, allowing the existence of protocols such $\mathrm{Clone}_B(\pi)$ that provide non-repudiability of messages despite the sender believing that they are taking part in a repudiable protocol π.

A point worth noting is that some mutually-authenticated protocols such as Signal, OTR, and TLS guarantee the authenticity of Alice's messages even when Bob's identity key is available to the adversary. In this case, an attacker can compromise Bob's device and carry out this attack *at any time*. This fact has been used to construct online attacks on deniability, such as in [34, §A]; they point out that an *online* trusted third party (or even SGX) can be used to obtain a non-repudiable proof of authenticity specifically for OTR and Signal.

Protocols such as DAKEZ [34] provide online deniability by allowing forgery by anyone holding *either* identity key. Such protocols resist our attack if Bob's identity key is generated or exported outside the TEE. Our attack is still applicable if the adversary can generate a new identity key inside a TEE and have it accepted by Alice [30], for example by corrupting Bob's key—thus forcing him to generate a new one—or compromising his device before he installs his messaging client. For this reason, such protocols are more resistant—though not invulnerable—to our attack.

Let us consider a concrete example of transforming the Signal protocol π_{Signal} to $\mathrm{Clone}_B(\pi_{\mathrm{Signal}})$. As described in Figure 1, this involves a regular Signal client on Bob's device augmented with a custom addition by the adversary that uses the TEE on Bob's device to produce an attestation of the output produced by the regular Signal client. This will convince Valerie that Bob received an authenticated message from Alice. The Signal protocol π_{Signal} does not interact with the TEE, and so this modification from π_{Signal} to $\mathrm{Clone}_B(\pi_{\mathrm{Signal}})$ is undetectable to Alice.

3.3 Practical attack

We have implemented this attack using remote attestations provided by Intel's SGX. We have produced an SGX enclave based on the *libsignal-protocol-c* [6] library to perform all session-related cryptographic operations; it produces all ephemeral keys, and the cryptographic state of the protocol never leaves the enclave in the clear. We then modify the third-party *signal-cli* Signal client to use this enclave in place of its own implementation of π_{Signal}.

Signal's key-establishment protocol at its most basic involves four key-pairs: one identity key-pair for each party—(A, g^A) and (B, g^B)—one ephemeral key-pair for the initiator—(a, g^a)—and one short-term pre-key pair (b, g^b) that is published by the recipient to allow asynchronous operation [25]. The root key is derived from these keys using a Diffie-Hellman key exchange, with the complication that several exchanges are computed simultaneously and combined according to $\mathrm{KDF}(g^{ab} \parallel g^{aB} \parallel g^{Ab})$. Reading or forging messages therefore requires at least one private key from each handshake; that is to say, at least one from $\{a, b\}$, one from $\{a, B\}$, and one from $\{A, b\}$.

An identity key is not enough to obtain the session key; this is necessary for forward secrecy. Deniability of the key exchange derives from the fact that possession of both ephemeral private keys suffices to obtain the session key, and thus anyone can forge a key exchange between any two parties. This feature is helpful for our attack, because if we know that an enclave has generated b and maintained its secrecy, then derivation of the key by any other entity requires the secret keys a and A—and unless Alice is compromised, only she will have access to A.

In addition to the normal processing that is performed by *libsignal-protocol-c*, the enclave constructs a transcript of the session, eventually producing an attestation to the entire transcript. This attestation proves to Valerie that Bob's secret ephemeral key b was secured by the TEE, and so that he cannot forge messages purporting to be from Alice.

An important point is that we do not require that the TEE maintain the secrecy of Bob's long-term identity key. The result is that the attack can take place at any time, for example following the compromise of a user's device, rather than at the time of key generation—this is the ideal situation for a remote attacker, as by compromising a victim's phone, this attack can be used to obtain a non-repudiable transcript of any messages sent to them from then on.

3.4 Other targets for attack

The attack described above is applicable not only to messaging protocols such as Signal and OTR, but to any system that provides authentication. This includes systems that involve a trusted third party, such as web-based messaging or email.

3.4.1 Web-based messaging. Consider web-based private messaging systems such as those by Facebook [5] and Twitter [31] that are accessed via server-authenticated TLS. Remote attestation can be used to obtain a transferable proof that, according to the service provider, a certain message was sent or received by the compromised user.

3.4.2 Local email. The attack generalizes to higher-level protocols that provide authentication in a non-cryptographic way, for example email between users of the same mailserver.

Take, for example, the mail system shown in Figure 2. A trusted mailserver accepts new mail via SMTP [19], and allows users to view and modify the contents of their mailboxes via IMAP [14] over TLS, and request a change of password over HTTPS. While email is in general vulnerable to forgery, a trusted mailserver can provide a secure messaging functionality to *local users.* The mailserver verifies user identities with a username and password over TLS; any mail received by SMTP purporting to be from a local user is rejected if the SMTP session is not authenticated in this way. Since SMTP is the only way to write to another user's mailbox, this provides a form of secure messaging.

If the IMAP session is protected by TLS, then a user Bob can prove the state of his mailbox to Valerie by using a TEE to obtain an attestation to the mailserver response over TLS. However, he cannot prove that a message in his mailbox came from another local user Alice, since the IMAP protocol allows anyone with Bob's password to write to the mailbox as well.

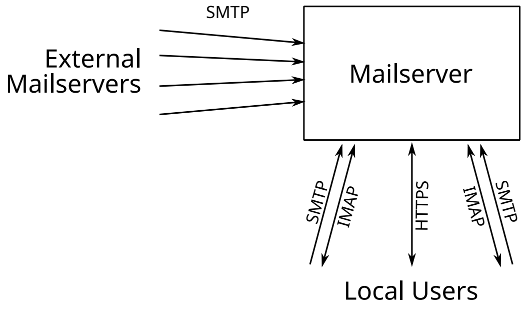

Figure 2: A mail system providing authenticated messaging between local users. Each local user accesses the server using a changeable password over TLS. Each user can append or delete messages from their own mailbox using password-authenticated IMAP over TLS. It is possible to append new messages to other users' mailboxes using SMTP, but a message purporting to be from a local user will only be accepted if the sender uses TLS and authenticates themselves using their password. Password changes can be made using HTTPS.

Suppose Bob sets his password using HTTPS and does not share it with anyone. He can then be sure that only he can access his mailbox and insert fraudulent messages; he keeps track of any messages that he inserts himself, and accepts as legitimate any messages from local users that have appeared in his mailbox since the password change, but which he did not add to the mailbox himself. Since the only other way that messages from a local user can be added to the mailbox is via authenticated SMTP, this protocol provides a secure messaging messaging functionality from local users to Bob. So long as the mailserver does not sign messages, such as with DKIM [21] signatures, this protocol is also deniable, since only Bob knows that he did not insert his own messages with IMAP.

Nevertheless, as an authenticated messaging protocol, we can attack this system by performing the protocol above inside a TEE. We have Bob's TEE change his password to a random value, known only to the TEE. From then on, Bob's account is only accessible via the TEE, which will attest only to messages that have appeared in the mailbox since the password change, but not those that Bob inserts or modifies via IMAP. The result is that Bob can prove to Valerie that Alice sent a supposedly-deniable message.

3.4.3 Voting systems. Though we focus on deniable messaging in this paper, the same cloning protocol allows Bob to convince Valerie of arbitrary protocol outputs, not just $\mathcal{F}_{\text{AUTH}}$ as we prove in Section 5.

This is particularly relevant to online voting; an important goal of online voting systems is to prevent vote-buying and coercion. This is accomplished in a number of ways, but in general uses a property known in the literature as *receipt-free-ness* [9].

Our attack applies to all systems that allow user devices to determine whether or not a final vote has been cast for a particular candidate (as opposed to one that requires outside information in order to decide, e.g. a randomized paper ballot). For the simplest electronic voting system—a web form with a list of candidates—an attestation to its submission over TLS will do. The exact form of this attestation depends on the system, but if a voter's device can ascertain their vote, it will in general be able to prove this to Valerie.

In cases where a voter can vote multiple times, the adversary must prevent the attested vote from being overridden after the fact. For example, if the system is completely online and allows login credentials—e.g. passwords and password-recovery settings—to be changed, the adversary can have the TEE lock the user out of the system after a single vote, thereby making it final.

4 MITIGATION

4.1 Regaining deniability with remote attestation

We have shown how remote attestation can be used to break deniability properties; in this section, we show how remote attestation can make authenticated protocols deniable once more.

We do this by including attestations in the protocol, such that Alice can refuse to send any messages to Bob unless he uses remote attestation to prove that he does not use his TEE to harm Alice's deniability.

Though disrupting individual realizations in this way does not protect against all adversaries, it can be used to prevent an attacker from using the TEEs already present in a compromised device; if the channel is bound to a particular piece of hardware, then the available TEEs will in some cases be known to Alice, allowing this attack to be prevented completely. Otherwise, an adversary might, for example, run an SGX enclave within a TPM-protected system, for example, and obtain a TPM-attestation to the output of the enclave, even though it is unattested by SGX.

The inability to counter a hardware-modifying attacker in this way is best explained by the following example: Bob might allow his part in the protocol to be performed by some generally-trusted human notary who attests to its output. Such social methods do not affect the protocol execution, and so this is an unsolvable problem in general.

If the recipient is not bound to a particular piece of hardware, all adversaries are hardware-modifying—a remote attacker cannot change Bob's hardware, but they can steal his key and run the protocol on a device over which they have arbitrary control. Any attestation-based countermeasure therefore requires that the channel be linkable to a specific device.

4.1.1 Complete protocol attestation. One approach is as follows: implement π—for concreteness, let us suppose that it is the Signal protocol—inside a TEE, but when Bob sends any public key to Alice, he obtains a remote attestation to the fact that the corresponding private key is protected inside the TEE, and sends this attestation alongside the public key. Before sending any messages using this new key, Alice can inspect the code run by Bob to verify that messages she sends will never be attested to, whether directly or indirectly. Alice is thereby assured of the deniability of her messages.

This countermeasure is straightforward, and has the peculiar advantage of being able to rule out even online attacks, since Alice can see that Bob has not modified his protocol implementation. The disadvantage is that the entire protocol implementation is within the trusted computing base of the system—in practice, this will mean trusting the developer, as proving the absence of even some covert, indirect form of attestation is a formidable task.

4.1.2 Protocol-independent countermeasures. As an alternative, we might consider an approach that requires only a very small piece of trusted code. Bob uses an TEE to obtain a remote attestation to the public key corresponding to a private key that is verified as being present in an unprotected location. This proves to Alice that Bob's device can perform a man-in-the-middle attack on the messages she sends, and therefore any attestation that Bob obtains on her messages will be insufficient to prove authenticity. This trusted functionality involves only a pointer check and a single elliptic-curve operation, and is therefore small enough that formal verification is realistic.

Another advantage of this method is that it does not require that every software developer produce their own trusted application; rather, the trusted part of the application is specific only to a cryptosystem, and thus with only a few such libraries it is possible to secure a wide variety of protocols.

The disadvantage of this approach is that the Bob cannot use his TEE to protect his own session keys; using a TEE to protect the entire protocol might allow greater security by keeping the its keys secure, but it is necessary to trust a large code-base. Conversely, using attestation to prove forgeability allows Bob to promise deniability using only a very small trusted application, but limits his ability to take advantage of the positive aspects of his TEE.

4.1.3 Adding attestations to the Signal protocol. The X3DH [25] Diffie-Hellman handshake used by the Signal protocol combines two Diffie-Hellman key pairs to provide both authentication and forward-secrecy. Rather than directly attesting to the non-protection of each secret key, we instead generate and attest an extra keypair at the beginning of each session. Then, in each handshake, the ephemeral Diffie-Hellman secret g^{ab} is instead replaced by $g^{ab} \parallel g^{aq} \parallel g^{bp}$, where p is generated and held by Alice's TEE, and q by Bob's TEE. At the beginning of each session, g^p and g^q are attested by Alice's and Bob's TEEs respectively, and sent to the other party. The secret keys p and q are confined to their respective TEEs. If one party does not have a TEE, then they do not send any g^p, and it is not included as input to the key derivation function.

For the 'large-TCB' countermeasure from Section 4.1.1, this occurs in the same TEE as the protocol itself. From Alice's perspective, since she keeps a secret, anyone else who obtains g^{aq} must know q. Bob's attestation proves to Alice that his q and the derived symmetric key k_{att} are confined to the non-message-attesting TEE that implements the protocol while keeping b secret. The same guarantee holds in reverse for Bob.

In the case of the 'small-TCB' countermeasure from Section 4.1.2, each enclave simply computes k_{att} and writes it to an unprotected region of memory. This ensures that Alice and Bob can use k_{att} to forge messages at will.

4.1.4 Defeating the TPM. We claimed before that an attestation-based countermeasure must individually prevent each TEE from attesting to the protocol output. Most systems with support for

SGX-based TEEs will also have a TPM, which can attest to the state of the entire system.

The TPM is relatively difficult to defeat in practice, owing to the difficulty of extending the measurement chain all the way to the application—an adversary must achieve this on *one* platform, while the defender must do so on *every* platform used to run the protocol. This prevents the use of countermeasures from Sections 4.1.1 and 4.1.2, which require attestation. The most practical means of defeating the TPM is therefore to prevent it from performing message attestations by breaking the chain of trust early in the boot process. At this point, remote attestation serves only to prove that a given value is present on this non-attesting system, allowing us to prove that the shared symmetric key exists outside a TEE. This approach has the rather substantial downside of preventing the use of the TPM elsewhere in the system, but it is currently the only viable means of defeating such an attack on the PC.

4.2 Regaining deniability by abandoning authentication

The fundamental problem that we demonstrate in this paper is that deniability and sender-authentication cannot coexist in a world with secure remote attestation. Thus, the only way to avoid non-repudiation cryptographically is to abandon sender-authentication.

This may seem like a drastic measure, but it is important to note that this does not mean the abandonment of message-authentication, but only *in-band* sender-authentication; that is, sender-authentication that occurs as part of the message-authentication protocol. Authentication performed outside the protocol, such as in-person verification of public keys, cannot be proven by an attestation, and can therefore be used to salvage deniability.

4.2.1 Linkable message authentication. An alternative to sender-authentication is to make messages linkable but not authenticated. Rather than having one long-term identity key that is used to communicate with all other users, and which can be used to prove the sender's identity, each user generates long-term keys that are unique to an individual session. The messaging protocol does not authenticate the sender of each message, but assures the recipient that all the messages in a session have come from the same sender; as a result, this is all that can be proven by attestation. In-person verification of the session keys can be used to obtain sender authentication, but this cannot be verified by the device, making it unattestable.

The Signal user interface has already moved in this direction, with each conversation having an individual 'safety number' that is the concatenation of the two users' long-term identity key fingerprints. The interface discourages the use of identity keys outside the context of an individual pair of users, by showing these two keys as a single large block of numbers, and so it is possible to switch to a linkability-only model without any user interface changes; a major downside is that the changeover must invalidate any in-person verifications already made, since the new 'per-pair' keys cannot be automatically connected to the current 'global' long-term identity keys, as otherwise the link can be established by attestation.

4.2.2 Practical difficulties. Such a model is far from foolproof. Some kind of per-user identifier will be needed, if only to make initial contact—in the case of Signal, this is a phone number. The discovery service that responds to this request must not authenticate the connection between users and keys, for example by responding using server-authenticated TLS, as then our attack can be used to obtain a statement that, according to the discovery service, a particular key belongs to a particular user. Despite this precaution, the discovery service can still map users to keys, forcing users to trust a central service to maintain the secrecy of their identities, and to trust themselves not to inadvertently link their own identity to the transcript, for example by disclosing secret information that only they know, which later becomes public.

A fully decentralized system such as Briar [4], in which users cannot communicate without prior verification, is more resistant to such attacks, but because in-band authentication methods such as the web of trust [27, §24.12] cannot be used, this would result in a return to the pre-Diffie-Hellman world of bilateral physical key exchange, a trade-off that few users would tolerate in practice.

4.3 Switching to online-deniable protocols

Online-deniable protocols such as RSDAKE [33] and DAKEZ and ZXDH [34] prevent online attacks on deniability; this means that we cannot use a TEE to mount an online attack 'offline' as shown in [34, §A.2] for Signal and OTR. They achieve this by allowing Bob to forge messages to his TEE using his identity key, something that is not possible with Signal or the current version of OTR (OTRv3).

As we show in Section 5, our attack applies to *any* purely cryptographic protocol implementing $\mathcal{F}_{\text{AUTH}}$. As we discussed in Section 3.2, in the case of DAKEZ and ZXDH, this requires that Bob's identity key be generated and confined within the TEE implementing $\text{Clone}_{\mathcal{B}}(\pi)$ [30]. This restricts the time window in which an attacker can compromise Bob's device without Alice detecting them by a key change.

The upcoming OTRv4 protocol will use the online-deniable DAKEZ and XZDH protocols [30]. Nevertheless, it is plausible that an attacker might convince Bob that his key material has become corrupted or otherwise needs to be regenerated, encouraging him to tell Alice that she need not be alarmed by the changed identity key. This therefore provides only partial mitigation, particularly against users lacking great discipline with respect to identity keys.

4.4 Setting correct expectations

Developers of messaging protocols may deem the mitigations above unaffordable. In that case we recommend that they make clear to their users the level of deniability that they can expect.

This is important because non-transferability is an major expectation of deniable protocols by users, who do not expect that their messages can be published and publicly verified; for example, the Signal website says the following [23] about deniability:

> One of OTR's primary features is a property called deniability. If someone receives an OTR message from you, they can be absolutely sure you sent it (rather than having been forged by some third party), but can't prove to anyone else that it was a message you wrote. This is a nice change compared to PGP signatures, for instance, where anyone who receives a

> PGP signed message can prove exactly who wrote it
> to anyone else.

This security property has motivated a number of high-value targets to switch to deniable messaging applications in place of email [17] following the *Podesta* email dump, in which a large number of emails were published to Wikileaks, many of them including signatures that can be used to verify their authenticity [3].

We have shown that the wide availability of hardware-supported attestation invalidates this expectation, and users facing such attacks may have to accept this risk if their application developers are unable to provide some mitigation.

5 SECURITY ANALYSIS

Deniability is an unusual property in that it requires that a certain attack be *possible*; as a result our attack is unusual in that it requires that the protocol $\text{Clone}_B(\pi)$ described in Figure 1 be *secure*. Conversely, the goal of our countermeasures—described in Section 4—is to change the original protocol so that any statement about it afterwards by either party is *vulnerable* to forgery.

Our analysis of the protocol $\text{Clone}_B(\pi)$ has two goals:

(1) To show that the attestation for the protocol cannot be forged—that is, a soundness proof to the benefit of Valerie.
(2) To show that Alice cannot distinguish between our protocol and the base protocol π.

Though we devote most of this section to proving the first point, the second is equally important. If Alice can detect that Bob is making a non-repudiable record of her statements, then she can abort.

5.1 Unforgeability

We begin by showing that the protocol is sound. Bob might try to frame Alice by sending Valerie an x that is different from what Alice really sent. Valerie has no contact with Alice, or even with the adversary during the execution of the protocol; in the most extreme case, Bob may not send his message (x, σ) to the verifier until years later. Our proof proceeds as follows:

(1) Use the game from Figure 3—taken from [8, Figure 1]—to infer the existence of a machine that has executed the protocol π_B and output the value m.
(2) Suppose π UC-emulates the functionality $\mathcal{F}_{\text{AUTH}}$.
(3) Construct a universal-composability experiment based on the following:
    - **Real World:** Execute $\text{Clone}_B(\pi)$, where Bob is incorruptible, and produces attestations using the functionality $\mathcal{F}_{\text{CERT}}$ from [12].
    - **Ideal World:** Simulate $\text{Clone}_B(\pi)$ for $\mathcal{A}$ by taking the simulator $\mathcal{S}$ from the UC-emulation proof for π, and augmenting it with extra calls to the adversary to simulate the attestation by Bob and the message from Bob to Valerie.
(4) These two worlds are indistinguishable until the end of execution of π_B, based on the definition of $\mathcal{S}$.
(5) After π_B, our simulator is an exact simulation of the real-world protocol, and thus the two worlds cannot be distinguished by $\mathcal{E}$.

In the real-world protocol described in Figure 1, the adversary hosts the TEE and sees its output, and so in this section we consider an authenticated message functionality $\mathcal{F}_{\text{AUTH}}$, rather than the secure message functionality $\mathcal{F}_{\text{SMT}}$ that also provides confidentiality.

```
prms ←$ $\mathcal{M}$.Init($1^n$)
$(P, L^*, l, n, \text{st}_{\mathcal{A}})$ ←$ $\mathcal{A}_1$(prms)
$P^*$ ← Compile(prms, $P$, $L^*$)
$\text{st}_V$ ← (prms, $P$, $L^*$)
/ The adversary produces $n$ attestations, possibly interacting with the TEE.
for $k \in [1 \ldots n]$
    $(i_k, o_k, \text{st}_{\mathcal{A}})$ ←$ $\mathcal{A}_2^{\mathcal{M}}(\text{st}_{\mathcal{A}})$
    $(o_k, \text{st}_V)$ ← Verify(prms, $l$, $i_k$, $o_k^*$, $\text{st}_V$)
    / The adversary loses if the attestations are invalid.
    if $o_k = \perp$
        return False
    fi
done
/ The adversary loses if the attestations are legitimate.
for hdl* : $\text{Program}_{\mathcal{M}}(\text{hdl}^*) = P^*$
    $(l'_1, i'_1, o'_1, \ldots, l'_m, i'_m, o'_m)$ ← $\text{Trace}_{\mathcal{M}_R}(\text{hdl}^*)$
    $T'$ ← filter[$l$]($\text{Trace}_{[\text{st};\text{Coins}_{\mathcal{M}}(\text{hdl}^*)]}$)
    if $T \sqsubseteq T'$
        return False
    fi
done
/ If the attestations are valid but not genuine, the adversary wins.
return True
```

Figure 3: The security game for Labelled Attested Computation, as given in [8, Figure 1]. The adversary interacts with a machine type $\mathcal{M}$, which may be any number of physical devices, and produces an attestation trace. The adversary wins if they succeed in obtaining a set of valid attestations that were not produced by a machine running the specified program.

We begin by constructing a model in the universal composability framework, using a security definition for remote attestation to infer the existence of a physical party running the protocol in question. Let us consider the computational framework from [8]; they describe a machine, deterministic except via a system call providing fresh random coins, that takes labelled inputs to produce optionally-attested outputs. Their definition of security has the adversary perform arbitrary interactions with the machine, producing a set of labelled input/attested-output pairs [8, Fig. 1]. The adversary wins if the attestations are valid, but no execution environment on the machine has legitimately generated the same input/output sequence.

If we suppose that the adversary cannot win with a non-negligible probability, then we know that, except with a negligible probability, that there is a machine that has executed π_B yielding output m. We introduce an incorruptible party into our real-world model that executes π_B and hands a signature to $\mathcal{A}$ to be given to V.

Lemma 5.1. *Suppose that Bob has access to a machine for which no adversary can win the remote-attestation security game from Figure 3 with non-negligible probability. Then, the real-world protocol* $\mathrm{Clone}_B\,\pi$ *is modelled in the universal-composabilty setting by Figure 4, noting that the additional incorruptible party executing* π'_B *physically exists in the form of an adversary-controlled* TEE.

Proof. By assumption, Bob cannot win the game from [8, Fig. 1] with more than negligible probability. We may therefore ignore the possibility that he will win the game, and focus on the ways in which he might lose.

In order for Valerie not to abort upon receiving the message from Bob, the attestation must be valid. We therefore need only consider the other failure condition for the adversary, namely that in which the machine has correctly executed the program $\mathcal{P}$, yielding the given output.

Because we know that a machine running this protocol physically exists somewhere, we may introduce one such machine as an incorruptible party π'_B to the protocol in Figure 4.

Since the attested value is, by definition, a message from some instance of π'_B, we model the attestation signature using Canetti's certification functionality $\mathcal{F}_{\mathrm{CERT}}$ [12]. This functionality allows us to ensures that a given message originated from π'_B, which is exactly the guarantee given by the attestation. Note that this only works because there is only a single attestation per protocol-execution, and thus the adversary cannot interleave messages from different protocol instances[2]. □

This model of our protocol is advantageous in that it allows us to construct our $\mathrm{Clone}_B(\pi)$-simulator by modification of an arbitrary simulator for π.

The isolation guarantees of the TEE are important here—in the UC model of the protocol, π'_B is executed by a distinct party. A perfectly isolating execution environment will make this party incorruptible, while a completely non-isolating execution environment—e.g. SGX operating in debug mode—manifests itself as a corruptible party.

Next we must specify the exact ideal-world functionality $\mathcal{F}_{\mathrm{CLONE}}$ that our protocol is to implement. This is shown in Figure 5, with reference to the ideal functionality $\mathcal{F}_{\mathrm{AUTH}}$ that is UC-emulated by the original protocol π. We take the original functionality $\mathcal{F}_{\mathrm{AUTH}}$ and have it ignore any messages from the verifier, but duplicate to Valerie any message sent to Bob.

Theorem 5.2 (Security of $\mathrm{Clone}_B(\pi)$). *Suppose the original protocol* π UC*-emulates* $\mathcal{F}_{AUTH}$*. Then,* $\mathrm{Clone}_B(\pi)$ UC*-emulates* $\mathcal{F}_{CLONE}$.

Proof. Since π UC-emulates $\mathcal{F}_{\mathrm{AUTH}}$, for any adversary $\mathcal{A}$ there must exist a block-box simulator $\mathcal{S}$ of π, such that the environment cannot distinguish between the following:

(1) The real-world protocol π being run with an adversary $\mathcal{A}$.
(2) The ideal-world functionality $\mathcal{F}_{\mathrm{AUTH}}$ being run against the simulator $\mathcal{S}$.

Our goal is to construct a simulator $\mathcal{S}'$ that cannot be distinguished from the real protocol $\mathrm{Clone}_B(\pi)$ by the environment.

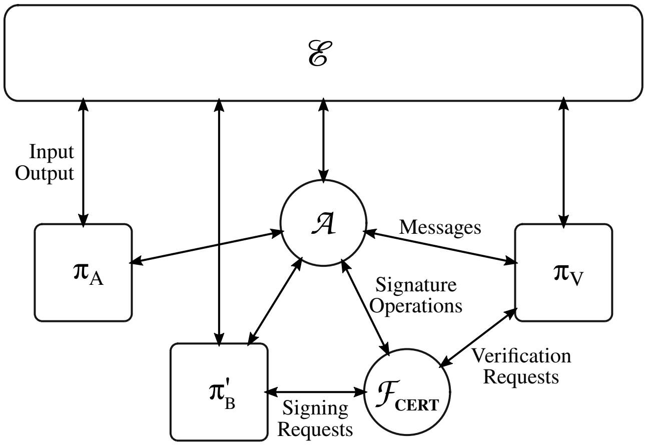

Figure 4: Hybrid model of our protocol in the universal composability setting. We model the remote attestation using the certification functionality from [12]—because π'_B performs only a single attestation, this models a remote attestation.

To do this, we augment $\mathcal{S}$ with an exact simulation of the extra messages provided by $\mathrm{Clone}_B(\pi)$. Note that $\mathrm{Clone}_B(\pi)$ is identical to π up until the attestation; therefore, we can use $\mathcal{S}$ to simulate this part of the protocol.

Each time $\mathcal{S}'$ receives a message M from $\mathcal{F}_{\mathrm{CLONE}}$, it does as $\mathcal{S}$ does until it reaches the end of the simulation of π.

At this point the environment cannot distinguish between the hybrid and ideal worlds, because the real-world execution is simply π, and this is simulated by $\mathcal{S}$.

We must now simulate the remainder of the protocol $\mathrm{Clone}_B(\pi)$, in which Bob's device attests to his output and sends it to Valerie. We perform a perfect simulation of the signature and verification using the contents m of the message leaked by $\mathcal{F}_{\mathrm{AUTH}}$: the behavior of π' at this point can be completely determined by the value of the message m leaked to the adversary.

This is a perfect simulation of the adversary's view of the protocol. Jointly with this, we consider the parties' outputs to the environment. Alice and Bob do not produce any outputs after they finish executing π, as in the ideal-world case. However, Valerie does produce an output at this point, if and only if the attestation she receives is accepted as valid. Otherwise, she aborts.

By the definition of $\mathcal{F}_{\mathrm{CLONE}}$, the message $(sid, \mathrm{Sent}, \mathrm{Alice}, \mathrm{Bob}, m)$ received by the adversary from the ideal functionality corresponds exactly to the input $(sid, \mathrm{Send}, \mathrm{Bob}, m)$ given to Alice as input by the environment. Therefore, the simulated protocol must—and in Figure 5, does—abort if and only if $\mathcal{A}$ responds negatively to the simulated verification request for the attestation of m. Otherwise the simulator may terminate, and Valerie will, by the definition of $\mathcal{F}_{\mathrm{CLONE}}$, output the correct value.

This is a perfect simulation of the real participants' behavior after the original protocol. The simulated protocol execution as a whole is therefore indistinguishable from a real protocol execution by the environment, and protocol thus UC-emulates $\mathcal{F}_{\mathrm{CLONE}}$. □

[2] This problem is quite easy to solve, even if this protocol does not require that we do so here. For example, a π'_B that performs several attestations might generate a random session identifier and include it in its attested data.

The ideal functionality $\mathcal{F}_{\text{AUTH}}$ adapted from [11].

```
When receiving $(sid, \text{Send}, R, m)$ from party $S$:
1 :  Send $(sid, \text{Sent}, S, R, m)$ to $R$.
2 :  Send $(sid, \text{Sent}, S, R, m)$ to $\mathcal{A}$.

Otherwise:
Ignore the message.
```

The ideal functionality $\mathcal{F}_{\text{CLONE}}$.

```
When receiving $(sid, \text{Send}, R, m)$ from $S \neq V$ such that $R \neq V$:
1 :  Send $(sid, \text{Sent}, S, R, m)$ to $R$.
2 :  Send $(sid, \text{Sent}, S, R, m)$ to $\mathcal{A}$.
3 :  Send $(sid, \text{Sent}, S, R, m)$ to $V$.

Otherwise:
Ignore the message.
```

The simulator $\mathcal{S}'$ for π in the $\mathcal{F}_{\text{CLONE}}$ model.

```
/ Simulate $\pi$.
1 :  Do as per $\mathcal{S}$ until its simulation of $\pi$ is complete.
/ Simulate the behavior of $\mathcal{F}_{\text{CERT}}$ during the attestation.
/ Let $m$ be the message from Alice received from $\mathcal{F}_{\text{CLONE}}$ during Step 1.
2 :  Send $(sid, \text{Sign}, m)$ to $\mathcal{A}$ from the simulated $\mathcal{F}_{\text{CERT}}$.
3 :  Upon receiving $(sid, \text{Signature}, m, \sigma)$ from $\mathcal{A}$ to $\mathcal{F}_{\text{CERT}}$,
     proceed as per Canetti's simulator of $\mathcal{F}_{\text{CERT}}$, shown in [12, Fig. 2].
/ Simulate the receipt of the attested output to $V$.
4 :  Upon receiving a message $(sid, x, \sigma)$ from $\mathcal{A}$ to $V$,
     send $(sid, \text{Verify}, x, \sigma)$ to $\mathcal{A}$ from the simulated $V$.
5 :  Upon receiving a message $(sid, \text{Verified}, x, r)$ from $\mathcal{A}$ to $\mathcal{F}_{\text{CERT}}$,
     if $x = m \wedge r \neq 1$ then abort.
```

Figure 5: The ideal functionality $\mathcal{F}_{\text{AUTH}}$ implemented by π, its variation $\mathcal{F}_{\text{CLONE}}$ implemented by $\text{Clone}_B(\pi)$, and the $\mathcal{F}_{\text{CLONE}}$ simulator $\mathcal{S}'$. We show that $\mathcal{S}'$ in the $\mathcal{F}_{\text{CLONE}}$ model is indistinguishable from the hybrid model of π in Figure 4. This definition captures the fact that we wish to provide the same functionality as $\mathcal{F}_{\text{AUTH}}$ to Alice and Bob, but to provide Valerie and adversary with Bob's output.

5.2 Undetectability

We finish by showing that the substitution of a protocol π by $\text{Clone}_B(\pi)$ is undetectable by Alice.

Theorem 5.3 (Indistinguishability by Alice). *Let π be a protocol that* does not *contain any calls to some* TEE *TEE($\mathcal{P}$) running the program $\mathcal{P}$ that implements* $\text{Clone}_B(\pi)$*.*

Then, the modified protocol $\text{Clone}_B(\pi)$ *is statistically indistinguishable by Alice from the original protocol π.*

Proof. We follow a hybrid argument. Noting that π does not use the trusted execution environment, and so Alice cannot detect changes in the derivation of messages to other parties, the following protocols are all perfectly indistinguishable from one another by Alice:

(1) $\text{Clone}_B(\pi)$
(2) *Protocol #1 with Valerie removed, along with the message from Bob to Valerie.*
Valerie does not send any messages to Alice or Bob, and so this change does not affect Alice's view.
(3) *Protocol #2 with Bob's attestation and* TEE *removed.*
The original protocol π_B does not use the TEE, and so its execution is not affected by removing it.
Bob's attestation occurs only after communication with Alice has finished; therefore, the protocol formed by step #2 is indistinguishable by Alice from the protocol formed by taking the same protocol and removing the attestation.

Protocol #3 is π, and thus $\text{Clone}_B(\pi)$ is indistinguishable by Alice from π. □

It is important to note that this holds true for *any* TEE with which the protocol π does not interact. This makes hardware-modifying adversaries particularly powerful.

Corollary 5.4. *A hardware-modifying adversary can perform this attack undetectably, even if Alice requires that Bob perform remote attestation.*

Proof. Let π be an arbitrary authenticated messaging protocol, where π_B may or may not include calls to a TEE. A hardware-modifying adversary Bob can add new forms of TEE to the system, and in particular, can nest the machine running π_B inside another TEE; for example, if π_B uses includes an SGX attestation, then Bob can run it within a TPM-measured application.

This means that a hardware-modifying adversary can always construct a protocol $\text{Clone}_B(\pi)$ that meets the requirements of Theorem 5.3, and hence any protocol implementing $\mathcal{F}_{\text{AUTH}}$ can be attacked without detection by such an adversary. □

6 UPGRADING DENIABLE AUTHENTICATORS TO SIGNATURES

Despite our offensive use of the protocol in Figure 1, non-repudiability is highly desirable in many other systems. For example, consider an automotive setting where a sensor containing a resource-constrained microcontroller unit (MCU) montitors airbag deployment. In case of an accident, the investigation process can benefit from an unforgeable report from the MCU indicating whether the airbag deployed. While the MCU can be equipped with hardware-protected secret key, it is too resource-constrained to sign every piece of data that it emits. However, a message authentication code (MAC) does not require much processing power. If the MCU shares a symmetric key with a TEE on the vehicle, which can verify the MAC, sign the message, and place it into the audit log. Shared symmetric keys between each sensor and the TEE, can be established either at the time of manufacture or each time the car is started. This approach will allow auditors to verify the provenance of data even from more limited sensors.

In general, this type of protocol allows an arbitrarily-authenticated message to be made non-repudiable. This is essentially a simple hardware security module, and we refer to the resulting signature as a *translated signature.*

While this is perhaps an obvious application of a TEE, we briefly discuss it for a number of reasons. First, such 'obvious' applications are often discussed but rarely rigorously analyzed [29]. Secondly, such a protocol is extremely similar to that described in Figure 1, but its requirements differ in a manner that contradicts the analysis in Section 5. Because of this, we leave its analysis for the extended version of this paper.

A practical translated signature protocol might involve the following:

(1) *During the setup phase,* Alice registers with the trusted application Bob and sets up a shared symmetric key, with Bob using remote attestation to show that this key is available only to the trusted application.
(2) Bob generates a signing key and enrols it with a registration authority—which can be either a separate entity, or part of the trusted application—to provide a binding between the signing key and Alice.
(3) *During the online phase,* Alice sends data to the server, authenticated with its shared symmetric key. Bob then signs the data with Alice's signing key, and sends the signature either to Alice or some other entity, such as an audit log.

The server-authentication process in step one violates the requirement, given in Figure 1, that π not use a TEE, but the resulting loss of indistinguishability is not a problem for this application.

In particular, when Alice uses this signature protocol, it is neither necessary nor desirable that she be unable to detect that she is taking part in something more than $\mathcal{F}_{\text{AUTH}}$. On the contrary: in many applications Alice must be certain that her signing key remains safely inside the TEE, used only to sign messages that she herself has authenticated.

7 RELATED WORK

Deniability has a long history in the cryptographic literature, and protocols such as Off-the-Record [10] and Signal [24] have been designed with the express goal of providing repudiability. These protocols are nonetheless vulnerable to *online* attacks, in which the verifier communicates with one of the parties during the protocol, and a number of protocols have been designed with this model in mind [16, 33, 34]. In a sense, our attack can be seen as running one of these online attacks locally inside a TEE, as foreshadowed by [34], who propose an attack similar to ours specifically against OTRv3 and Signal using a trusted third party. However, unlike previous work, our attack is more general and applies to even online-deniable protocols and higher-level protocols as shown in Section 3.

In fact, the Town Crier protocol [35] is quite similar to that that we describe in Section 6. They use a trusted execution environment to produce a certification that an input to a smart contract originated from a trusted feed at a certain time. The implications for deniability were not realized at the time, and so they did not consider the possibility that such a protocol can be used adversarially, nor the question of whether a feed operator might prevent their data from being used in such a way.

The TLS-SIGN [28] and TLS-N [26] extensions to TLS provide non-repudiation by signing all or part of the data stream, but require that the server be modified to provide a digital signature. This is effective where the server is willing to cooperate, but cannot be used in practice unless the server operator is willing to expend effort in order to make their responses non-repudiable. Unlike our approach, it can not be used by a client to hold a server accountable against its will.

Other approaches to server-supported signatures have been proposed that provide some level of accountability on the part of the server. For example, [7] combines a normal signature scheme with a hash chain; the client releases a hash pre-image for each signing request, which the server signs along with the data. For the server to produce extra signatures beyond those requested by the client it must re-use an element of the hash chain, but the discovery of distinct signatures that include the same hash pre-image provides cryptographic evidence of the server's misbehavior, meaning that it can be held accountable if such signatures are found in the wild.

8 DISCUSSION

That remote attestation can compromise deniable messaging protocols is somewhat obvious in hindsight; nevertheless, even [35], which makes use of the phenomenon in a fundamental way, fails to anticipate the far-reaching implications of a protocol that obtains a transferable authenticator from a deniable protocol.

This attack is highly practical with existing hardware; the SGX-based realization of this attack is mitigated somewhat by the need for an Intel-whitelisted signing key in order to run an SGX enclave with memory protection, but for a well-resourced attacker this is unlikely to pose an enormous obstacle. Though more difficult to use, TPM-based attestation is available to anyone, and so restrictions on the use of SGX do not prevent this attack in general.

8.1 TEE-based countermeasures

The need for a TEE in order to retain existing protocol guarantees has a number of implications. First, if we are to provide both sender authentication and deniability in the same protocol, the use of a TEE is mandatory. That is to say, there is no purely cryptographic method by which deniability can be achieved without sacrificing authenticity. Secondly, the asymmetric nature of the threat puts the defender at a distinct disadvantage. Now that TEEs capable of remote attestation are available, to be assured of deniability it becomes necessary for *everyone* to abandon either purely-cryptographic protocols or machine-verifiable sender authentication. Even supposing that an attestation-based defense is viable, there will be a long transitional period during which the relevant hardware and software is not sufficiently ubiquitous as to allow users to refuse communication with those that fail to provide the proper attestations.

In addition, the TEE-based countermeasures that we describe in Section 4.1 require complete enumeration of the TEEs present in the system; on mobile platforms this might be realistic if it is possible to obtain an attestation to the model of the device. However, such an enumeration will be most reliable in an organizational setting, where the capabilities of issued devices can be exactly known.

The effect on the software ecosystem is also substantial: with applications generally being unable to use a TEE without some kind of commercial relationship with its designer, it will no longer be possible for users to arbitrarily modify their messaging applications, as is the case in the open-source world today. This disadvantage might be greatly ameliorated by allowing the general public to access TEE functionality in some limited way—while TEE vendors are hesitant to allow free access so because of malware concerns, defenses that we propose in this Section 4.1 will be equally effective if they allow untrusted code read-only access. This will allow arbitrary applications to perform meaningful attestations without the risk of the platform being misused to produce un-analyzable malware.

8.2 Trust in the TEE

Another important point is that it is not necessary to have universal trust in the TEE: only Valerie needs to trust the TEE, and so the fact that some user might hold that a TEE can easily be physically attacked is irrelevant—if their local journalists trust the TEE, then that is enough for an adversary seeking to provide a credible email dump.

Backdoor-ed TEEs are also irrelevant to our protocol in at least one important case—if we suppose that the politicians and officials of any given nation use devices whose TEEs are manufactured in their own country, then even if a backdoor is present, will not be accessible by a foreign adversary to be of use for forgery. Incriminating messages obtained by a foreign power can therefore still be verified: while they might conceivably be forged, a well-designed backdoor will allow only the target nation to forge attestations, and thus the victim's nation cannot deny the attested messages by blaming a backdoor-ed TEE.

8.3 Comparison to forensic methods

Finally, we take a moment to compare our attack with the methods normally used in criminal investigations.

A judge can rely on many different kinds of non-cryptographic evidence: Bob might testify under oath, or his device might be examined by a forensic technician who is trusted to give honest evidence.

What these methods have in common with our attack is that some party provides a link in the chain of trust between Alice's input to the protocol and the value that the judge accepts. However, there is an important difference in that these methods rely on the existence of such a party who is directly involved in the case. When an attack is made by an untrustworthy adversary—for example, a hostile government—then it is unlikely that the adversary will be able to obtain such a signed—and so transferable—statement from a source trusted by the public.

Our attack demonstrates that such a trustworthy party is now widely available in the form of a TEE with remote attestation capabilities. Fortunately, protocol builders seeking to achieve a practical form of deniability can choose to mitigate this attack by the methods discussed in Section 4.

8.4 Responsible disclosure

We informed the Signal and OTR developers of this work on 30 April 2018.

The Signal developers have responded that they "don't consider this a protocol issue, or an issue that affects deniability in practice."

The OTR developers have responded that, motivated by similar concerns, in the upcoming OTRv4 protocol they have already decided to use online-deniable key-establishment protocols, a mitigation we discuss in Section 4.3.

9 CONCLUSION

In this work, we show how a TEE can be used to convert a deniable authenticated channel into a non-repudiable one. While this ability can been used legitimately for such purposes as implementation of remotely-accessible HSM [22] or the injection of data into smart contracts [35], that one can do so surreptitiously has far-reaching implications that are obvious only with the benefit of hindsight.

We have shown that this applies to any protocol implementing an authenticated message functionality. Protocol designers therefore face a difficult choice: abandon either unconditional deniability or some level of authenticity, or incorporate trusted execution environments into their protocols. Compatibility concerns render the latter unrealistic in the short-term, leading us to the unfortunate conclusion that even off-line deniable communication is no longer practical for most users without in-person verification.

More generally, remote attestation changes adversary models in a non-trivial way. In some ways, an adversary using remote attestation is weaker, because it cannot arbitrarily deviate from the protocol specification. But when selecting an adversary model for deniable protocols, power is weakness and honesty strength. With remote attestation capabilities widely available, it is necessary to reconsider whether existing protocols provide the same security guarantees under this adversary model. In the case of deniable messaging, the answer is no.

ACKNOWLEDGMENTS

We thank Chris Brzuska for helpful discussion, as well as Jian Liu and Andrew Paverd for reviewing previous versions of this paper. We thank the developers of Signal and OTR for their prompt responses and discussion.

This work is supported in part by Intel (ICRI-CARS) and by the Academy of Finland (grant 309195).

REFERENCES

[1] 2009. *ARM Security Technology: Building a Secure System using TrustZone Technology* (3 ed.). White paper. ARM. https://www.arm.com/products/security-on-arm/trustzone

[2] 2014. *Intel Software Guard Extensions Programming Reference.* Technical Report. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf

[3] 2016. DKIM Verification. (2016). https://wikileaks.org/DKIM-Verification.html

[4] 2018. Briar: Secure messaging, anywhere. (2018). https://briarproject.org/ accessed 2018-04-29.

[5] 2018. Messenger. (2018). https://www.messenger.com/ accessed 2018-05-03.

[6] 2018. *Signal Protocol C Library.* Code. https://github.com/signalapp/libsignal-protocol-c commit 9e10362fce9072b104e6d5a51d6f56d939d1f36e.

[7] N. Asokan, Gene Tsudik, and Michael Waidner. 1996. Server-Supported Signatures. In *ESORICS'96: 4th European Symposium on Research in Computer Security (Lecture Notes in Computer Science)*, Elisa Bertino, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo (Eds.), Vol. 1146. Springer, Heidelberg, 131–143.

[8] Raad Bahmani, Manuel Barbosa, Ferdinand Brasser, Bernardo Portela, Ahmad-Reza Sadeghi, Guillaume Scerri, and Bogdan Warinschi. 2017. Secure Multiparty Computation from SGX. In *FC 2017: 21st International Conference on Financial Cryptography and Data Security (Lecture Notes in Computer Science)*, Aggelos Kiayias (Ed.), Vol. 10322. Springer, Heidelberg, 477–497.

[9] Josh Cohen Benaloh and Dwight Tuinstra. 1994. Receipt-free secret-ballot elections (extended abstract). In *26th Annual ACM Symposium on Theory of Computing.* ACM Press, 544–553.

[10] Nikita Borisove, Ian Goldberg, and Eric Brewer. 2004. Off-the-record communication, or, why not to use PGP. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*. https://doi.org/10.1145/1029179.1029200

[11] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 136–145.

[12] Ran Canetti. 2003. Universally Composable Signatures, Certification and Authentication. Cryptology ePrint Archive, Report 2003/239. (2003). http://eprint.iacr.org/2003/239.

[13] Irin Carmon. 2013. How we broke the NSA story. *Salon* (2013). https://web.archive.org/web/20130615053746/http://www.salon.com/2013/06/10/qa_with_laura_poitras_the_woman_behind_the_nsa_scoops/singleton/ 2013-06-10.

[14] Mark Crispin. 2003. Internet Message Access Protocol - Version 4rev1. RFC 3501. (March 2003). https://doi.org/10.17487/RFC3501

[15] Tim Dierks and Christopher Allen. 1999. *RFC 2246 - The TLS Protocol Version 1.0.* Internet Activities Board.

[16] Yevgeniy Dodis, Jonathan Katz, Adam Smith, and Shabsi Walfish. 2009. Composability and On-Line Deniability of Authentication. In *TCC 2009: 6th Theory of Cryptography Conference (Lecture Notes in Computer Science)*, Omer Reingold (Ed.), Vol. 5444. Springer, Heidelberg, 146–162.

[17] Mara Gay. 2017. Political world embraces encrypted-messaging app Signal amid fears of hacking. *The Wall Street Journal* (2017). https://www.wsj.com/articles/political-world-embraces-encrypted-messaging-app-amid-fears-of-hacking-1485492485 2017-01-27.

[18] ISO/IEC 11889:2015 2015. *Trusted Platform Module Library.* Standard.

[19] John C Klensin. 2008. Simple Mail Transfer Protocol. RFC 5321. (Oct. 2008). https://doi.org/10.17487/RFC5321

[20] Hugo Krawczyk. 1996. SKEME: A versatile secure key exchange mechanism for Internet. In *Proceedings of the Symposium on Network and Distributed System Security*. https://doi.org/10.1109/NDSS.1996.492418

[21] Murray Kucherawy, Dave Crocker, and Tony Hansen. 2011. DomainKeys Identified Mail (DKIM) Signatures. RFC 6376. (Sept. 2011). https://doi.org/10.17487/RFC6376

[22] Arseny Kurnikov, Andrew Paverd, Mohammad Mannan, and N Asokan. 2018. https://arxiv.org/abs/1804.08569. (2018). https://arxiv.org/abs/1804.08569

[23] Moxie Marlinspike. 2013. Simplifying OTR deniability. (2013). https://signal.org/blog/simplifying-otr-deniability/ accessed 2018-05-01.

[24] Trevor Perrin and Moxie Marlinspike. 2016. *The Double Ratchet Algorithm.* Standard. Open Whisper Systems. https://signal.org/docs/specifications/doubleratchet/

[25] Trevor Perrin and Moxie Marlinspike. 2016. *The X3DH Key Agreement Protocol, Revision 1.* Standard. Open Whisper Systems. https://signal.org/docs/specifications/x3dh/

[26] Hubert Ritzdorf, Karl Wüst, Arthur Gervais, Guillaume Felley, and Srdjan Capkun. 2017. TLS-N: Non-repudiation over TLS Enabling - Ubiquitous Content Signing for Disintermediation. Cryptology ePrint Archive, Report 2017/578. (2017). http://eprint.iacr.org/2017/578.

[27] Bruce Schneier. 1996. *Applied Cryptography.* Wiley.

[28] Ahmed Serhrouchni and Ibrahim Hajjeh. 2006. Intégration de la signature numérique au protocole SSL/TLS. *Annales Des Télécommunications* 61, 5–6 (2006), 522–541. https://doi.org/10.1007/BF03219921

[29] Yogesh Swami. 2017. SGX Remote Attestation is not Sufficient. Cryptology ePrint Archive, Report 2017/736. (2017). http://eprint.iacr.org/2017/736.

[30] OTRv4 team. 2018. Personal communication. (2018).

[31] Twitter. 2018. About Direct Messages. (2018). https://help.twitter.com/en/using-twitter/direct-messages accessed 2018-05-03.

[32] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. 2015. SoK: Secure Messaging. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 232–249. https://doi.org/10.1109/SP.2015.22

[33] Nik Unger and Ian Goldberg. 2015. Deniable Key Exchanges for Secure Messaging. In *ACM CCS 15: 22nd Conference on Computer and Communications Security*, Indrajit Ray, Ninghui Li, and Christopher Kruegel: (Eds.). ACM Press, 1211–1223.

[34] Nik Unger and Ian Goldberg. 2018. Improved Strongly Deniable Authenticated Key Exchanges for Secure Messaging. (2018). Issue 1. https://doi.org/10.1515/popets-2018-0003

[35] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town Crier: An Authenticated Data Feed for Smart Contracts. In *ACM CCS 16: 23rd Conference on Computer and Communications Security*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 270–282.