Secure Remote Attestation with Strong Key Insulation Guarantees

Deniz Gurevin, Chenglu Jin, Phuong Ha Nguyen, Omer Khan, Marten van Dijk

Abstract—Secure processors with hardware-enforced isolation are crucial for secure cloud computation. However, commercial secure processors have underestimated the capabilities of attackers and failed to provide secure execution environments capable of protecting sensitive information against side channel attacks. Remote Attestation protocols based on traditional signature schemes are not secure under side channel attacks anymore since their secret keys can be leaked. Previously, Key-Insulated Schemes (KIS) have been introduced to mitigate the damage caused by secret key exposure in cryptosystems by breaking the lifetime of secret keys into independent sessions. KIS protect the security of all other sessions if any session keys are compromised, however, provide no security guarantees for a compromised session. We introduce a new cryptographic primitive called One-Time Signature with Secret Key Exposure (OTS-SKE), which ensures no one can forge a valid signature of a new message or nonce even if all secret session keys are leaked. OTS-SKE enables us to sign attestation reports securely under a powerful adversary who can observe all digital states in secure enclaves through side channel attacks. We also minimize the trusted computing base by introducing a secure co-processor that is only responsible for key generation into the system. Our experiments show that the signing of OTS-SKE is faster than KIS as well as Elliptic Curve Digital Signature Algorithm (ECDSA) used in Intel SGX.

Index Terms—Remote Attestation, One Time Signatures, Secure Processor Architecture

1 Introduction

Sensitive computations are being increasingly deployed on shared pay-per-use infrastructures that leverage economies of scale and drive down costs. These shared services expose software systems and even physical hardware components to emerging security vulnerabilities. This trend has led to the deployment of trusted execution environments (TEE), more recently coined as *Confidential Computing* platforms. Driven by the security challenges, the semiconductor industry has adopted a paradigm shift in security as most major instruction set architectures have added support for confidential computing. Examples include Intel SGX, TDX, AMD SEV, and ARM CCA that aim to enable application isolation technology, *enclaves* [1], [2], [3], [4], [5].

In general, the secure processor technology [1], [6], [7], [8], [9], [10], [11], [12], [13] is based on hardware isolation and remote attestation (RA) principles. Hardware isolation allows one to run a code snippet in an enclave that is isolated from the OS and other enclaves with the goal of keeping its internal computations private. Besides being able to execute code in a trusted execution environment that guarantees privacy, a remote user also needs to be able to verify whether a computed result originated from the executed code. RA is based on digital signature schemes that sign and bind a computed result to the enclave code that produced it, along with the processor identity. A remote user needs remote attestation to verify the results produced by such an enclave. Usually, an *asymmetric key crypto-system* is adopted for attestation purposes, so that an attestation can be performed using the private key in an enclave isolated platform, and then verified by a remote user using the corresponding public key that is known to the verifier.

Unfortunately, hardware isolation that RA relies on for its security has shown to be elusive. The enclave platform, where the remote attestation is performed, itself may have vulnerabilities and can possibly be exploited through its own I/O interactions. With the developing capabilities of adversaries and side channel attacks, vulnerabilities of processors executing enclaves continuously keep getting exploited, leaking private digital state (including private keys). A recent survey [14] has shown that Intel SGX has been susceptible to a wide range of attacks [15], [16], [17], [18], [19]. We may conclude that **hardware isolation as is implemented today for executing enclave code cannot guarantee privacy**. An RA scheme that relies on hardware isolation to hide its attestation keys will suffer from side channel attacks, which can leak these secret keys when they are used inside enclaves. In fact, any internally computed enclave value may potentially leak, and we cannot make any solid privacy guarantee. Therefore, we cannot trust the current state-of-the-art processor technology's RA.

Key-Insulated Schemes (KIS) have been introduced to mitigate the damage caused by secret key exposure in public cryptosystems [20], [21]. KIS breaks the lifetime of the secret key into multiple sessions and updates it at the beginning of every session. KIS typically has an architectural design with a *user* and a *base*. The secret keys are held in shares by the user and its base, and all secret session keys correspond to one universal public key. Hence, the public key does not need to be updated frequently, while the secret session keys are refreshed for each session. A *perfectly* key-insulated scheme protects the security of *all other sessions* if any session keys are compromised [20]. However, KIS signature schemes leave a loophole for attackers to exploit, i.e., the KIS signatures provide *no security guarantees for a compromised session*. If a session key is leaked, the attacker

- *D. Gurevin and O. Khan are with the Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT 06269, USA. E-mail: deniz.gurevin, khan@uconn.edu*
- *C. Jin and M. Dijk are with CWI, Amsterdam, The Netherlands. M. van Dijk is affiliated with the Vrije Universiteit van Amsterdam and University of Connecticut. E-mail: chenglu.jin, marten.van.dijk @cwi.nl*
- *P. H. Nguyen is with eBay, San Jose, CA, USA. E-mail: phuongha.ntu@gmail.com*

can use the leaked key to forge valid signatures of any messages specific to the compromised session. One can use a synchronization mechanism between the signer and verifier to synchronize a clock and enforce the expiration of a session key so that a signature can only be deemed valid within a time window. Nevertheless, having a synchronization mechanism still does not completely protect the security of the leaked session since the attacker can still forge valid signatures within the time window.

We introduce a new cryptographic protocol, **One-Time Signature with Secret Key Exposure (OTS-SKE)** that *guarantees the security of all sessions, including the compromised session*. In OTS-SKE, each secret session key is unique to *both* the message and a fresh nonce sent by the user. Hence, even if a session key is leaked, it can only be used to sign the message-nonce pair that the session was scheduled to sign. Signing a different message or nonce using the leaked session key will result in a failure in signature verification. Figure 1 depicts our solution, where a one-directional memory component is added that ensures one-time use of session keys. The OTS-SKE scheme signs each user's message with a secret session key tailored for the user's nonce and message. Unlike KIS, OTS-SKE does not require a synchronization mechanism between the signer and the user to enforce the expiration of a session key since the session key automatically expires with each usage. An adversary who is able to leak a session key cannot use this key to sign another message to forge a signature since the key does not match with another message-nonce pair. Consequently, the forged signature fails during the verification by the user. Therefore, OTS-SKE severely limits the exposure to the adversary.

Both KIS and OTS-SKE protocols require their private key generation to be secure, and the master secret key that is responsible for session private keys should be kept hidden. Since the enclave processor takes inputs from a user, it cannot be isolated from the adversary. Therefore, as shown in Figure 1, we introduce a truly isolated piece of hardware that is not affected by any adversary – a secure co-processor that is dedicated to generating secret keys. This is accomplished by enforcing a unidirectional interaction between the KeyGen co-processor and the enclave processor, and hence the co-processor only generates fresh secret keys and does not take any inputs. The session keys in both protocols can be observed by the adversary once they enter the enclave processor. However, in OTS-SKE, they are discarded as soon as they are used. This requires the co-processor to generate new session keys at the rate with which remote attestations are being requested.

The secure key generation co-processor generates a set of secret keys sk_t for each session t, and stores these subkeys in a special memory component, called *One-Directional Memory* (ODM) [22], [23] as highlighted in Figure 1. The signing enclave then selects a subset sk_x of these keys based on a message including the user's nonce to create a unique session key that binds to the user and the message. All secret keys that once leave the one-directional memory are exposed to the adversary, and they are used for signing the attestation report. The output sk_x returned by the ODM is essentially already a signature due to its unique dependence on the combination of a message and nonce. Therefore, the role of the signing function is to combine the subset sk_x of subkeys for that session to generate a shorter signature. The

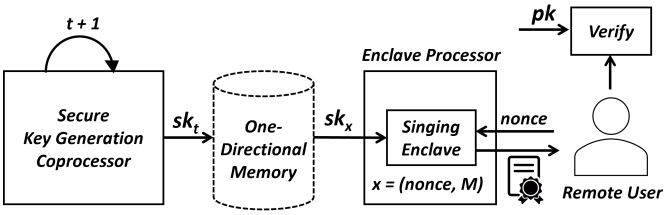


Fig. 1. Overview of the proposed RA protocol. A remote user makes an attestation request sending a random nonce to the enclave processor. The KeyGen co-processor generates and transfers session keys using the unidirectional interface enabled by the One-Directional Memory. The RA/signing enclave creates a RA signature with session key sk_x which is read out as a *subset* of the key material sk_t most recently generated by the KeyGen co-processor; sk_x corresponds to the user's nonce and message. The remote user uses the public key to verify the RA signature.

signature produced by the signing enclave is verified by the user using a single universal public key, the message, and its nonce. To ensure one time use, ODM implements a special mechanism for the used and unused session keys: after reading a subset of the subkeys chosen by the enclave processor from the one-directional memory, all subkeys of that session are automatically erased. This way, the unused session keys are not exposed to the adversary. However, even though the used session keys are exposed, the adversary is unable to forge another valid signature since these keys are unique to each usage.

In this paper, we make a key observation that KIS is not able to protect remote attestations against impersonation attacks that involve an adversary that leaks the session keys and use them to forge signatures. We introduce the **OTS-SKE cryptographic primitive that improves security over KIS by protecting against an adversary who is capable of leaking all current and past secret session keys and launching signature forgery attacks using them**. Building upon the proposed OTS-SKE construction, we propose a remote attestation scheme and demonstrate its effectiveness and performance compared to the KIS-based remote attestation scheme for secure processors. We also compare the performance of the OTS-SKE and KIS schemes against Elliptic Curve Digital Signature Algorithm (ECDSA) deployed in existing commercial secure processors. Our evaluation shows that the performance of OTS-SKE is constrained by the rate of generating private keys. We have implemented the key generation on an Intel machine, as well as an area and power-optimized ARM processor. The key generation on the secure co-processor executes in parallel with the signing. Therefore, the performance overhead of the proposed protocol can maintain high throughput if the key generation processor keeps up with the expected signing rate.

1.1 Paper Outline

In Section 2, we give an overview of the remote attestation protocol implemented in commercial secure processor architectures, and their shortcomings in the presence of side-channel attacks. Motivated by these security challenges, we define a strong adversary that is able to leak secret keys. Section 3 introduces a baseline system implementation of the KIS-based RA protocol for secure processor architectures and discusses its security challenges. Section 4 extends KIS

and introduces the OTS-SKE protocol for secure processor architectures that is secure in the presence of the adversary. Finally, Section 5 evaluates the performance of the RA protocols.

2 Motivation and Adversarial Model

Today's secure processors from Intel and AMD support authentication with a remote user [2], [19]. For instance, in Intel SGX the RA flow starts with a remote user sending an attestation request, with a nonce to guarantee freshness, to the SGX platform. The SGX application receives this request and forwards it to its application enclave, which generates a local attestation report that includes the enclave's measurement and the nonce. The Quoting Enclave (QE) then verifies the local attestation and signs it with a *secret attestation key* that is provided by Intel. This signed report is called a *Quote* and can then be verified by the remote user using the public key. Intel SGX uses an ECDSA-based attestation, which allows third parties to build their own non-Intel attestation infrastructure. ECDSA uses 256-bit Elliptic Curve secret and public key pairs.

In all commercial secure processors, remote attestation relies on the usage of a *single* private signing key that is the root of trust. To secure the storage and use of these keys, the secure processors rely on hardware isolation principles and access control mechanisms. While theoretically intact, these hardware isolation primitives are not sufficient to prevent private data from leaking through side channels, i.e., other sources of information that can still be observed and used by a malicious adversary to extract sensitive information. In fact, in recent years, it has been shown that commercial secure processors are vulnerable against a wide range of side channel attacks [14], [17], [18], [19]. **We may conclude that the private keys that are used for attestations leak due to side channel attacks.** We can no longer only rely on hardware isolation for the maintenance of the private keys.

In order to provide a secure remote attestation protocol in secure processor architectures, we have to consider an adversary with all known and unknown side-channel attack capabilities. We assume an adversary that can observe and leak all digital secrets in the enclave processor where the RA enclave resides, including the secret signing keys for attestation. In other words, we do not trust the confidential computing offered by secure processor architectures as secret values inside enclaves leak through side channels. On the other hand, we trust the integrity of the code executed inside the enclave, i.e., we assume that the enclave processor keeps on functioning according to its specification which includes under which conditions (verified by hardware checks) an executing process (OS or enclave code) can access or manipulate data flow. Therefore, we consider a passive adversary that cannot tamper with computations in the processor but can observe them. Let us consider an adversary that

- can compromise and alter the OS, run its own enclave code, and can execute or interact with instantiations of the RA enclave,
- can observe all digital state of the enclave processor, which includes all intermediate digital values computed by the RA enclave as well as all digital storage together with register values, permanent storage, and fused (endorsement) keys,
- cannot tamper with the enclave processor's functioning in that its specification cannot be circumvented, in particular, the adversary is not physically present or has inserted hardware Trojans (as a consequence the adversary cannot circumvent the enclave processor's specified hardware checks in order to tamper with values computed inside the RA enclave or stored in the on-chip digital storage),
- cannot tamper with or observe the secret computations within the physically isolated key generation co-processor,
- and cannot tamper with or observe the stored values in the one-directional memory before they leave the one-directional memory.

These capabilities give the adversary the opportunity to steal digital keys and perform impersonation attacks. We argue that one can only achieve secure remote attestation if we can design a protocol that is intact in the presence of the proposed adversary.

3 KIS-based Remote Attestation

In order to mitigate secret key exposure, Key Insulated Schemes (KIS) have been introduced in literature [20], [21]. KIS implements a public cryptosystem by generating a public key along with a master secret key which is stored on a secure device. The lifetime of the secret key is divided into distinct periods of time, called *sessions*. A secret key for each session is generated from the master secret key, and then used for signing on an insecure device where the key exposure may occur. At the end of the session, the secret session key is discarded and replaced with a new session key. This makes it possible to refresh a potentially leaked secret key periodically, without the need to replace the public key. Because each secret session key is independent of one another, KIS reduces the damage caused by secret session key leakage.

In the remainder of this section, we describe the KIS-based RA in further detail. We start by giving the definition of a Key-Insulated Signature Scheme based on the scheme introduced in [21].

A KIS $\mathcal{S}$ consists of three procedures[1]

$$\mathcal{S} = (\textsc{KeyGen}, \textsc{Sign}, \textsc{Verify}) :$$

Key generation. Based on a security parameters λ, KeyGen generates a public key pk together with session secret key sk_i and auxiliary variables aux_i for each session $i \in \{0, \ldots, N-1\}$. We have

$$(pk, \{sk_i, aux_i\}_{i=0}^{N-1}) \leftarrow \textsc{KeyGen}(\lambda).$$

1. A typical KIS signature scheme consists of five procedures: key generation, device (base) key update, user key update, signing, and verification [21]. Because KIS schemes store and manage the secret keys of the device (base) and the user separately, they have separate procedures to generate the initial secret keys (the key generation procedure in [21]) and to update the device key and the user key. For the simplicity of our discussion and comparison with our scheme, we merge the device/user key update procedures into the key generation procedure in our discussion, such that the definition of KIS will be consistent with the one we propose for OTS-SKE.

Signing. SIGN takes as input the session id i with session secret key sk_i and auxiliary variable aux_i. Together with a message $M \in \{0,1\}^n$ as input SIGN produces a signature σ,

$$\sigma \leftarrow \text{SIGN}(sk_i, aux_i; M).$$

Verification. VERIFY outputs

$$\{\textbf{true}, \textbf{false}\} \leftarrow \text{VERIFY}(pk, i; \sigma, M)$$

for a signed message (σ, M) for session id i. Notice that the same public key pk is used for all sessions.

3.1 Design Requirements

Secret Key Generation. KIS requires its key generation to be on a physically secure device, i.e., the future/unused session keys should not be exposed to the adversary. Since the enclave processor where the RA report is signed cannot be trusted, the key generation must be offloaded to a completely isolated hardware that is only responsible for refreshing session keys. It is important to minimize the attack surface for key generation by removing any user-level communication channels. This is done by having a physically isolated key generation processor, that is separated from the enclave processor. The physically isolated co-processor generates secret signing keys at runtime that are concurrently used by the remote attestation enclave.

Local Attestation. In practice, an application enclave first uses local attestation to a quoting enclave that is responsible for the remote attestation protocol with the client. For this reason, we need to rely on secure local attestation in the presence of the adversary who can steal any master key of the local attestation mechanism and use this to remotely circumvent the application enclave and impersonate its identity. One solution is to implement local attestation as a physical authentic channel between enclaves. Here, the channel is hardware isolated such that the messages transmitted over the channel cannot be tampered with, and the source/authenticity of messages cannot be modified [11]. As another option, adopted in this paper, the application and remote attestation enclaves can be combined together in one single enclave such that a physical authenticated channel is inherently present. Here, each application enclave implements its own remote attestation. This discards the use of a quoting enclave (with higher permissions as in Intel SGX) which implements remote attestation for all application enclaves. Instead, each application enclave uses its own remote attestation code as a wrapper around the application enclave itself. In what follows, RA enclave should be interpreted as the application enclave with its own remote attestation wrapper code.

3.2 KIS-based RA Protocol

A remote attestation wrapper (RAWrapper) is merged with the application enclave (AppEnc). We want to show that AppEnc can have its computed result signed by RAWrapper for a remote user. Figure 2 depicts the solution. There are three main functions for the KIS scheme: KEYGEN, SIGN and VERIFY. The secure isolated co-processor implements KEYGEN to generate a sequence of session keys. The SIGN functionality is implemented by the RAWrapper, and the remote user uses VERIFY to verify the result sent by the RAWrapper.

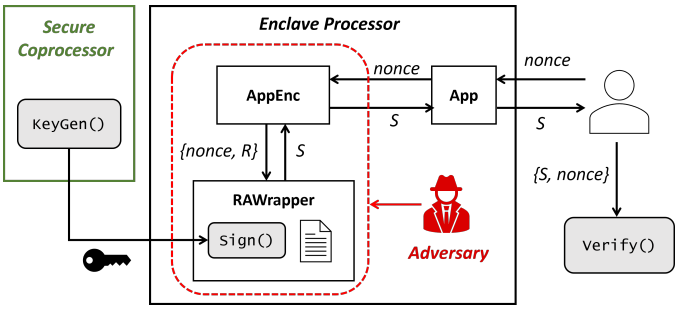


Fig. 2. KIS-based RA protocol. The remote user starts the RA process by sending a random nonce $nonce$. The AppEnc forwards this request along with its computed result R to RAWrapper. In order to retrieve a signing key, RAWrapper computes a message M that takes the hash of the AppEnc's measurement along with the result R and the user's $nonce$. RAWrapper uses the current session key that is provided by the Secure Coprocessor to create a signature and sends it back to the remote user, who can then verify the signature using the public key.

During the initialization phase of the RA protocol, the secure key generation co-processor bootstraps and uses KEYGEN to generate a public key pk and initializes the default state of the session counter i to 0. The public key must bind to the identity of the processor.[2] During the initialization of RAWrapper it takes this pk from the key generation co-processor and stores precomputed values that it can use during runtime for signing. The pk is known by the remote user and used during attestation verification. After the initialization, the essential components of the RA protocol are set up.

Algorithm 1 describes the signature generation process for the RA protocol. KEYGENPROCESSOR is responsible for the key generation module while RAWRAPPER(.) handles the RA requests from the remote user and is responsible for signature generation.

Key Generation. In the secure coprocessor, if the session key has expired (i.e., the time elapsed since the last session t is greater than the predefined session period ΔT), then the session counter i is incremented and published (Line 3–4). The coprocessor then starts pregenerating the next session's secret key sk_{i+1} along with the auxiliary information aux_{i+1} and sends the updated key to the RAWrapper (Line 5–6).

Signing. The signing is performed by the RAWrapper which resides in the insecure enclave processor. RA starts with RAWrapper receiving a signing request from the remote user along with its random nonce. The RAWrapper receives a result R from an AppEnc and combines AppEnc's measurement MR_{app} with R and $nonce$ to create a unique message, $M = \text{HASH}(\text{MR}_{app}, R, nonce)$ (Line 11–13). RAWrapper receives the current session secret key and its

2. During bootstrapping, this public key should leave the secure co-processor in a one-directional way and must be released to the clients securely. In other words, no adversary should be present during the initialization phase. In order to achieve this, the initialization mode of KEYGEN can for example be done on the manufacturer side (Intel) and the manufacturer can then certify the public key and serve as a third party that knows that the processor is associated with the specific public key.

Algorithm 1 KIS-based Remote Attestation

We assume a bilinear-map based KIS $\mathcal{S} = (\text{KEYGEN}, \text{SIGN}, \text{VERIFY})$. KEYGEN is implemented in a secure isolated co-processor and SIGN is performed in the RAWrapper. In the initialization phase, pk is generated and published, and the session counter variable $i = 0$ is initialized in the isolated secure co-processor. sk is updated periodically, i.e. when the time elapsed since the last session (t) is greater than the pre-defined key renewal period (ΔT).

1: **procedure** KEYGENPROCESSOR
2: **if** $t \geq \Delta T$ **then**
3: i += 1
4: Publish i
5: Send (sk_i, aux_i) to RAWrapper
6: $(sk_{i+1}, aux_{i+1}) \leftarrow$KEYGEN($\lambda$)
7: **end if**
8: **end procedure**
9: **procedure** RAWRAPPER
10: **while true do**
11: Pop a signing request from the queue
12: $(R, nonce, \text{RemoteUser})$
13: $M = \text{HASH}(\text{MR}_{app}, R, nonce)$
14: $(sk_i, aux_i, i) \leftarrow$ KEYGENPROCESSOR
15: $key \leftarrow (sk_i, aux_i)$
16: $\sigma = \text{SIGN}(key; M)$
17: $S \leftarrow (i, \sigma, \text{MR}_{app}, R)$
18: **send** S to RemoteUser
19: **end while**
20: **end procedure**

Algorithm 2 Request and Verification

We assume the remote user knows pk (as part of a certificate issued by a trusted CA) of the KIS $\mathcal{S} = (\text{KEYGEN}, \text{SIGN}, \text{VERIFY})$ used by AppEnc with RAWrapper.

1: **procedure** REQUESTANDVERIFY
2: **send** random $nonce$ to AppEnc
3: **receive** S from AppEnc
4: $(i, \sigma, \text{MR}_{app}, R) \leftarrow S$
5: $M = \text{HASH}(\text{MR}_{app}, R, nonce)$
6: **return** VERIFY$(pk, i; \sigma, M)$
7: **end procedure**

session ID from the KEYGENPROCESSOR, and uses it to sign M (Line 14–16). Finally, RAWrapper sends the session counter i and the resulting signature S consisting of σ, measurement of AppEnc, and the result R to RemoteUser (Line 17–18).

Verification. On the remote user's side (explained in Algorithm 2), after selecting a $nonce$ and making an RA request to RAWrapper with it, the user receives the created signature $S = (i, \sigma', \text{MR}_{app}, R)$ for the i-th session (Line 2–4). After computing M (Line 5), it can verify the validity of its signature using the public key pk and session counter[3] i (Line 6).

3. This includes verifying whether counter i corresponds to a ΔT time window not too far in the past.

3.3 Shortcomings of KIS-based RA Protocol

If the KIS-based RA protocol fails to prevent an adversary from stealing a session key, then the adversary can forge a valid signature within the same session. Because a single secret key is used multiple times for different users in a session (i.e., a time window), the key is not unique to the user's message. Therefore, an adversary who leaks the secret key in a session, can create his own message and sign it to create a valid signature for that session. To limit the adversary's capability to leak the secret key, the session length in KIS can be shortened to enforce the expiration of a session key more frequently to limit the damage of a compromised session. However, this still requires a reliable synchronization mechanism implemented between the user and processor to ensure freshness of the session key.

In the next section, we introduce One-time Signature with Secret Key Exposure (OTS-SKE) to make the remote attestation resilient to signature forgery attacks. The objective is to strengthen the security of KIS-based remote attestation in secure processors.

4 OTS-SKE REMOTE ATTESTATION PROTOCOL

In the proposed OTS-SKE scheme, an attacker *cannot* forge a valid signature of a *new* message for *any session* even if *all session keys* are leaked, while key-insulated schemes provide no security for the compromised sessions. Given this, the next sections describe the definition of OTS-SKE, the construction and implementation of this new signature scheme, and how it is contextualized in state-of-the-art secure processor architectures.

4.1 OTS Scheme with Secret Key Exposure

The idea is to have (1) one (universal) public key that can be used to verify all session signatures, (2) each session generates at most one signature with its own secret session key that is unique to a random nonce sent by the remote user and the message to be signed, and (3) this unique session key is exposed to the adversary for free. Since the session key is unique to the message, the key cannot be used to sign any other messages. Also, the key cannot be used to sign messages for other users due to the use of a random nonce.

An OTS-SKE scheme $\mathcal{S}$ consists of three procedures

$$\mathcal{S} = (\text{KEYGEN}, \text{SIGN}, \text{VERIFY}) :$$

Key generation. Based on a security parameters λ, KEYGEN generates a public key pk together with session secret keys

$$sk_i = \{sk_{i,j}\}_{j=0}^{q-1}$$

and auxiliary variables aux_i for each session $i \in \{0, \ldots, N-1\}$ and a-priori fixed parameter q. We have

$$(pk, \{sk_i, aux_i\}_{i=0}^{N-1}) \leftarrow \text{KEYGEN}(\lambda).$$

Signing. SIGN takes as input the session id i with session secret key sk_i and auxiliary variable aux_i. Together with a message $M \in \{0,1\}^n$ as input SIGN produces a signature σ,

$$\sigma \leftarrow \text{SIGN}(sk_i, aux_i; M).$$

The computation of SIGN is split in three steps:

1) We have a keyed pseudo random permutation $\mathrm{PRP}(key; x)$ which, for each key, is a bijective mapping from strings $x \in \{0,1\}^n$ to $\{0,1\}^n$. We also have an injective mapping ϕ from $\{0,1\}^n$ to subsets of $\{0,\ldots,q-1\}$ (here, $q \geq n$). SIGN first selects a random key and computes the subset
$$I = \phi(\mathrm{PRP}(key; M)) \subseteq \{0,\ldots,q-1\}.$$
2) SIGN extracts a corresponding subset of the i-th session secret key:
$$sk_{i,I} = \{sk_{i,j}\}_{j\in I}.$$
3) SIGN uses $sk_{i,I}$ together with aux_i and input message M to produce a signature σ'. In order to make the dependence on the subset of the session key explicit, we write
$$\sigma' \leftarrow \textsc{Sign}'(sk_{i,I}, aux_i; M).$$
SIGN returns $\sigma = (\sigma', key)$.

Verification. VERIFY outputs
$$\{\mathbf{true}, \mathbf{false}\} \leftarrow \textsc{Verify}(pk, i; \sigma, M)$$
for a signed message (σ, M) for session id i. Notice that the same public key pk is used for all sessions.

4.1.1 Correctness and Security Definition

We define correctness and security of the OTS-SKE scheme even if the adversary has the knowledge of subsets of session keys.

Correctness. OTS-SKE scheme $\mathcal{S}$ is correct if for all $\sigma \leftarrow \textsc{Sign}(sk_i, aux_i; M)$ we have $\mathbf{true} \leftarrow \textsc{Verify}(pk, i; \sigma, M)$.

Security. Even if an adversary has knowledge of subsets of session keys
$$\{sk_{i,I_i}\}_{i=0}^{N-1}$$
together with auxiliary information $\{aux_i\}_{i=0}^{N-1}$, the adversary cannot impersonate a signature for some session with id i^* for a new message that has not yet been signed in session i^*. This security notion is formalized by GameOTS-SKE for $\mathcal{S}$ as the following security game:

- **Setup**: The challenger runs KEYGEN which returns
$$(pk, \{\{sk_{i,j}\}_{j=0}^{q-1}, aux_i\}_{i=0}^{N-1}).$$
The challenger gives pk as well as $\{aux_i\}_{i=0}^{N-1}$ to the adversary.
- **Query**: The adversary adaptively issues a sequence of messages M_i at most one message for each session id i. The challenger computes
$$I_i = \phi(\mathrm{PRP}(key_i; M_i)) \text{ and } sk_{i,I_i} = \{sk_{i,j}\}_{j\in I_i}$$
for random key_i.
The challenger gives the extracted information sk_{i,I_i} with key_i to the adversary (as soon as M_i is received).
Notice that the adversary can use this information to sign message M_i for session i by applying SIGN′. This may lead to multiple signatures for M_i (since SIGN′ may use fresh randomness for each signature generation). However, no signatures for other messages $\neq M_i$ for session id i can be forged if the following Guess does not succeed.
- **Guess**: The adversary selects a session number $i^* \in \{0,\ldots,N-1\}$ which refers to the session for which the adversary will want to forge a signature: The adversary outputs a signed message (σ, M^*) for session i^* such that $M^* \neq M_{i^*}$
The adversary wins the game if the signature verifies, that is,
$$\mathbf{true} \leftarrow \textsc{Verify}(pk, i^*; \sigma, M^*).$$

In this game, the adversary, denoted by $\mathcal{A}$, is called an OTS-SKE-EUF-CMA (OTS-SKE Existential UnForgeability under Chosen Message Attack) adversary.

If $\mathcal{A}$ wins GameOTS-SKE with probability $\geq \epsilon$ in time $\leq T$, then we call $\mathcal{A}$ a (T, Q_P, ϵ)-OTS-SKE adversary for $\mathcal{S}$, where Q_P is the maximum number of queries allowed to be made by $\mathcal{A}$ to a PRP oracle in GameOTS-SKE. We say scheme $\mathcal{S}$ is (T, Q_P, ϵ)-secure against OTS-SKE-EUF-CMA attacks if no (T, Q_P, ϵ)-OTS-SKE adversary exists.

4.1.2 Bilinear-Map based OTS-SKE Construction

We introduce a new bilinear-map based construction that realizes a correct, secure OTS-SKE scheme $\mathcal{S}$. We begin with introducing the following definitions:

Bilinear map. Let $\mathbb{G}$ be a bilinear group of prime order p and g be a generator of $\mathbb{G}$. Here, size p of $\mathbb{G}$ is determined (by some functional relation) by the security parameter λ of the to-be-explained constructions. Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ be a bilinear map, i.e., we have the following properties

- *Bilinear* : For all $x, y \in \mathbb{G}$ and all $a, b \in \mathbb{Z}$,
$$e(x^a, y^b) = e(x, y)^{ab}.$$
- *Non-degenerate* : $e(g, g) \neq 1$.

For practical usage the bilinear map should be efficiently computable. The above properties can be realized by the modified Weil pairing based on supersingular curves.

OTS-SKE scheme. Below we describe our OTS-SKE scheme
$$\mathcal{S} = (\textsc{KeyGen}, \textsc{Sign}, \textsc{Verify}) :$$

Key generation. We use parameters $q = tn$ and represent index $tj + b \in \{0,\ldots,q-1\}$ as the pair (j, b). KEYGEN sets parameters, computes the public key, and all secret keys
$$(pk, \{\{sk_{i,j,b}\}_{j=0,b=0}^{n-1,t-1}, aux_i\}_{i=0}^{N-1}) \leftarrow \textsc{KeyGen}(\lambda)$$
as follows:

- $(p, \mathbb{G}, \mathbb{G}_1, e, g, g_2) \leftarrow IG(1^\lambda)$ where λ is the security parameter and algorithm IG generates a suitable mathematical structure for our signature scheme. g and g_2 are generators of $\mathbb{G}$ and $\mathbb{G}_1$, respectively.
- Randomly generate $\alpha \in \mathbb{Z}_p^*$ and set $g_1 = g^\alpha$. Define $F(i) = g_1^i h$ where h is a random number chosen from $\mathbb{G}$. Note that $F : \mathbb{Z}_p \to \mathbb{G}$.
- Generate N secret keys $\{sk_i\}_{i=0}^{N-1}$ with auxiliary information $\{aux_i\}_{i=0}^{N-1}$ as follows:
$$sk_i = \{sk_{i,j,b}\}_{j=0,b=0}^{n-1,t-1}$$

with

$$sk_{i,j,b} = g_2^{\alpha} F(it^n + bnt^j)^{r_i} v_{i,j} \text{ and } aux_i = g^{r_i}, \quad (1)$$

where r_i is a random number chosen from $\mathbb{Z}_p$ and $v_{i,j} = g^{\beta_{i,j}}$, where $\beta_{i,j}$ are random numbers from $\mathbb{Z}_p$ such that $\sum_{j=0}^{n-1} \beta_{i,j} = 0$, or equivalently, $\prod_{j=0}^{n-1} v_{i,j} = 1$.

- Parameters $pk = (p, \mathbb{G}, \mathbb{G}_1, e, g, g_1, g_2, h)$ are made public and secret keys $\{sk_i\}_{i=0}^{N-1}$ are kept private. The auxiliary information $\{aux_i\}_{i=0}^{N-1}$ is kept at the signer but is not kept secret (it can be accessed by anyone who wants to). Random numbers $\{r_i\}$ and $\{v_{i,j}\}$ are deleted.

We notice that KeyGen can be implemented in KEYGENPROCESSOR using an update rule as in [24] to create a continuous stream of session keys.

A deviation from [24] is the secret sharing mechanism based on the $\{v_{i,j}\}$, which role will become clear in the security analysis and allows us to achieve resistance against OTS-SKE-EUF-CMA attacks.

Signing. We compute $B = \sum_{j=0}^{n-1} b_j t^j$ with $0 \leq b_j < t$ as the pseudo random permutation output $B = \text{PRP}(key; M)$ for a random key. We define subset $\phi(B) = \{tj + b_j\}_{j=0}^{n-1}$ of $\{0, \ldots, q-1\}$ for $q = tn$. We represent its elements by the pairs (j, b_j). We produce

$$\sigma' \leftarrow \text{SIGN}'(\{sk_{i,j,b_j}\}_{j=0}^{n-1}, aux_i; M),$$

which signs a message $M \in \mathbb{G}_1$ as follows:

- Compute

$$\begin{aligned} sk_i &= \prod_{j=0}^{n-1} sk_{i,j,b_j} = \prod_{j=0}^{n-1} g_2^{\alpha} F(it^n + b_j nt^j)^{r_i} v_{i,j} \\ &= g_2^{n\alpha} \left(\prod_{j=0}^{n-1} F(it^n + b_j nt^j) \right)^{r_i} \\ &= g_2^{n\alpha} (g_1^{it^n + B} h)^{nr_i} = (g_2^{\alpha} F(it^n + B)^{r_i})^n. \end{aligned}$$

- Return signature $\sigma = (\sigma', key)$ for

$$\begin{aligned} \sigma' &= (y, z) \\ &= (aux_i, sk_i) \\ &= (g^{r_i}, (g_2^{\alpha} F(it^n + B)^{r_i})^n). \end{aligned}$$

Verification. VERIFY$(pk, i; \sigma, M)$ with $\sigma = (\sigma', key)$ verifies signature $\sigma' = (y, z)$ for message M, where σ' is generated during the i-th session:

- Compute $B = \text{PRP}(key; M)$.
- The signature verifies if and only if

$$e(g, z) = e(g_1, g_2^n) \times e(y, (g_1^k h)^n) \text{ with } k = it^n + B.$$

Correctness. The correctness of the scheme follows from $g_1 = g^{\alpha}$, $y = g^{r_i}$, $z = (g_2^{\alpha} F(k)^{r_i})^n$, and

$$\begin{aligned} & e(g_1, g_2^n) \times e(y, (g_1^k h)^n) \\ = \; & e(g^{\alpha}, g_2^n) \times e(g^{r_i}, (g_1^k h)^n) = e(g^{\alpha}, g_2)^n \times e(g^{r_i}, g_1^k h)^n \\ = \; & (e(g^{\alpha}, g_2) \times e(g^{r_i}, g_1^k h))^n = (e(g, g_2^{\alpha}) \times e(g, (g_1^k h)^{r_i}))^n \\ = \; & (e(g, g_2^{\alpha} (g_1^k h)^{r_i}))^n = (e(g, g_2^{\alpha} F(k)^{r_i}))^n \\ = \; & e(g, (g_2^{\alpha} F(k)^{r_i})^n) = e(g, z). \end{aligned}$$

Security. For any (T, Q_P, ϵ)-OTS-SKE adversary, for $\mathcal{S}$ with N sessions, there exists an algorithm that solves CDHP (Computational Diffie-Helman Problem) in $\mathbb{G}$ (elliptic curve) in expected time $\leq T(Q_P + 1)N/\epsilon$. The details of the security proof are provided as an appendix in the supplementary material.

4.2 Design Requirements

Based on our adversarial setting, we have the following requirement: we have to prevent the leakage of future, past and current secret session keys. Leakage of future secret session keys can be prevented by using an isolated secure key generation co-processor that is also required by KIS as described in Section 3.1. However, additionally, the past keys must be protected to prevent impersonation attacks by an adversary who is able to observe session keys once they enter the enclave processor. Hence, the signing key must be used only *once*, and the remote user must be able to verify its freshness by generating a session key that is unique based on her random nonce.

This is achieved by generating and storing in *One-Directional Memory* (ODM) a session key as a sequence of secret subkeys by using the strongly isolated key generation processor. Based on a random nonce received from the remote client (for proving freshness) and based on the to be signed message, the RA enclave/wrapper computes a nonce which it forwards to the ODM. The ODM combines and maps the nonce and the measurement of the RA enclave (includes the application code) to a subset of subkeys which are read and given to the RA enclave. By its specification, the ODM erases the rest of the sequence. The RA enclave combines the selected subset and creates a single unique session key, which is used by the RA enclave for signing. The adversary can observe this subset of keys, but despite leaking them, he cannot forge a signature since in our construction the selected subset of subkeys returned by the ODM already represents an unforgeable binding to the measurement of the RA enclave, the message to be signed, and the random nonce from the remote client.

4.3 Proposed Remote Attestation Protocol

The OTS-SKE based RA protocol follows the same building blocks as the KIS-based RA protocol described in Section 3.2. However, the one-directional memory is included as a buffer between the secure co-processor and the enclave processor for providing the secret session keys as shown in Figure 3. Different from KIS, now the secure co-processor generates and stores multiple subkeys per session in the ODM. Upon the RAWrapper's read request with its message x that includes the remote user's nonce, only a subset of secret session keys are selected corresponding to x and the rest are automatically erased.

The initialization phase follows the same procedure of KIS-based RA described in Section 3.2 where pk is generated and published, and the session counter i is initialized as 0. Algorithm 3 describes the working phase of the RA protocol. EODMEM(.) is implemented as an enclave call that handles the ODM's read requests and implements its access control mechanism.

Key Generation. KEYGENPROCESSOR increments the session counter i and generates the complete set of secret keys

Algorithm 3 OTS-SKE based Remote Attestation

We assume a OTP-SKE-EUF-CMA secure bilinear-map based OTS-SKE scheme $\mathcal{S} = (\text{KeyGen}, \text{Sign}, \text{Verify})$ where Sign is defined by procedure Sign′. KeyGen is implemented in a secure isolated co-processor and Sign is performed in the RAWrapper.

1: **procedure** KeyGenProcessor
2: $i = i + 1$
3: $key = (\{sk_{i,j,b}\}_{j=0,b=0}^{n-1,t-1}, aux_i) \leftarrow$KeyGen$(\lambda)$
4: Publish $\{aux_i, i\}$
5: Store $sk_i = \{sk_{i,j,b}\}_{j=0,b=0}^{n-1,t-1}$ in OD memory Mem
6: **end procedure**
7: **procedure** EODMem(x)
8: Compute $I = \phi(\text{Hash}(x, \text{MR}_{caller}))$
9: Read $y = (Mem_j)_{j \in I}$
10: Erase $(Mem_j)_{j \notin I}$
11: **return** y
12: **end procedure**
13: **procedure** RAWrapper
14: **while true do**
15: Pop a signing request from the queue
16: $(R$, RemoteUser,nonce$)$
17: $M = \text{Hash}(\text{MR}_{app}, R)$
18: $x = \text{Hash}(nonce, M)$
19: $\{aux_i, i\} \leftarrow$ KeyGenProcessor
20: $\{key_I\} \leftarrow$ EODMem(x)
21: $k \leftarrow$ Combine key_I, aux_i
22: $\sigma' = \text{Sign}'(k; M)$
23: $S \leftarrow (i, \sigma', \text{MR}_{app}, R)$
24: **send** S to RemoteUser
25: **end while**
26: **end procedure**

sk_i along with the auxiliary information aux_i (Line 2–3). It publishes aux_i with the session i, and it stores the set of subkeys that constitute session secret key $sk_i = \{sk_{i,j,b}\}_{j=0,b=0}^{n-1,t-1}$ in the one-directional memory (Line 4–5).

EODMem(.) is implemented as an ECall (enclave call) that handles the read requests for the one-directional memory. It computes the hash of input x, concatenated with the caller enclave's measurement MR_{caller} to compute the set $I = \phi(\text{Hash}(x, \text{MR}_{caller}))$ to extract a subset of keys from memory that is related to I (Line 8). Here, we define

$$\text{PRP}(i; M) = \text{Hash}(\text{Hash}(nonce_i, M), \text{MR}_{app}),$$

where $nonce_i$ is the nonce received from RemoteUser for session i. Regarded as a function of M a collision resistant hash function $\text{Hash}(nonce_i, M)$ cannot be distinguished from a pseudo random permutation with non-negligible probability. Therefore, we may use this for our PRP and fit the definition of the OTS-SKE scheme. As soon as the subset of secret keys y is extracted, the rest of the secret keys that are not included in subset I are erased from the one-directional memory, and y is returned to the caller (Line 9–11).

An example of how this process works is demonstrated in Figure 3. I is generated to map x to n t-ary symbols ($t = 3$ and $n = 4$ in the example). This implies that OTS-SKE key generation produces $nt = 12$ subkeys in a single session. The ODM selects exactly one subkey out of t subkeys for each j where $j = \{0, 1, ..., n-1\}$.

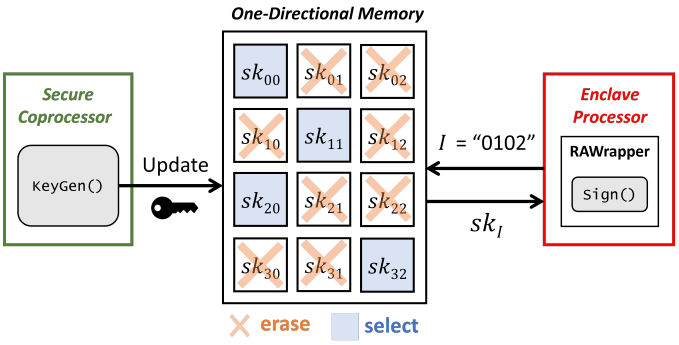


Fig. 3. The one-directional memory (ODM) stores the complete set of keys for a session i, $sk_{j,b}$ where $j \in \{0, 1, 2, 3\}$ and $b \in \{0, 1, 2\}$, generated by the secure key generation co-processor. When the RAWrapper makes a read request to the one-directional memory by sending its input $x = \text{Hash}(nonce, M)$, ODM extracts a unique subset of keys related to the input x, and immediately erases the keys that were not included in the subset. RAWrapper combines this subset of keys and uses it to create a signature and sends it back to the remote user, who then verifies the signature using the public key.

Signing. Remote attestation session starts with RAWrapper receiving a signing request. The RAWrapper receives a result R from an AppEnc, which needs to be signed for a remote user. RAWrapper combines AppEnc's measurement with R to create a unique message, $M = \text{Hash}(\text{MR}_{app}, R)$. After this, the random nonce, that is received as a part of the RA request, is used to compute the input $x = \text{Hash}(nonce, M)$ (Line 14–18). Meanwhile, RAWrapper receives the current session's auxiliary key along with the session counter i from KeyGenProcessor. In order to read the subset of secret keys from the one-directional memory, it makes the EODMem(x) call and it reads the secret key subset that is related to the input x (which is unique to $nonce$ and the message M) from the one-directional memory. It then combines this unique subset of secret keys, along with the auxiliary key aux_i to generate the signing key k, which is used to sign the message M (Line 19–21). Finally, RAWrapper sends the session counter i and the resulting signature S consisting of σ', measurement of AppEnc and the result R to RemoteUser (Line 22–24). The subkeys in the ODM are overwritten by the secure coprocessor in the next session.

Verification. On the remote user's side, after selecting a $nonce$ and making a RA request to RAWrapper with it, the user receives the created signature $S = (i, \sigma', \text{MR}_{app}, R)$ for the i-th session. The user uses Verify to verify the signature S. If verified, then the remote user knows that R was indeed created by AppEnc at the enclave processor: The chain of trust shows that the signature was created by RAWrapper, which only signs messages M that are a hash of MR_{app} with R.

4.4 Security of OTS-SKE based RA Protocol

In OTS-SKE, even if a current session key leaks, it cannot be used to impersonate a signature for the current session. OTS-SKE uses one-time signatures, whereas KIS uses a time window: During the time window, new signatures can be impersonated, as many as feasibly possible. OTS-SKE based RA protocol prevents an adversary from forging a

signature of a malicious result R for a remote user, even if the adversary leaks all the session keys. This is because each session key is unique to the user's message M. If the adversary observes and leaks the current or a past session's secret key, the observed session key will always be unique to another user's message (that includes the hash of the application enclave's report and the user's nonce together). The adversary can create his own message M^*, however, the stolen session key will not match the message, and the forged signature will fail during verification.

Moreover, the adversary cannot learn the unused portion of the session subkeys in the ODM and use them to forge a valid signature in that session. Even if the adversary runs the RAWrapper itself, it only learns at most one subset of keys $\{sk_{i,j}\}_{j\in I}$ and aux_i for a session id i, because once a subset is used for signing, i is incremented. If the adversary attempts to read the one-directional memory, the returned subset of keys depends on the adversarial enclave's measurement through a hash evaluation (Line 10 in EODMEM). Because of the hash collision resistance, the adversary cannot use EODMEM to learn a set of subkeys that fits MR_{app}. If the adversary observes the message M as supplied by the remote verifier, then he learns the related subset of keys of session i and can create other σ' for the same message M and session id i. However, as previously explained, he cannot generate a signature for a new malicious message M^*. A new malicious message M^* corresponds to a different subset of the session key indicated by a set I^*.

The security guarantee of the OTS-SKE scheme shows that the adversary cannot successfully forge a signature for M^*. We conclude that under the adversary our scheme offers secure RA.[4]

5 Implementation & Evaluation

In this section, we give the details of our implementation and performance analysis for our proposed protocol. We show the timings for key generation, signing, and verification phases of our proposed RA attestation.

5.1 Experimental Setup

All experiments are conducted on an Intel Xeon Gold 5218 CPU with 2 sockets, each supporting 16 cores, running at 2.3 GHz and using Ubuntu 18.04 operating system. For the parallel execution of the key generation module of the OTS-SKE-based RA protocol, the pthread library and g++ compiler (v 6.4.1) with the -O3 optimization flag are used.

The bilinear map based OTS-SKE scheme is implemented using the open-source MIRACL Multiprecision Integer Cryptographic Library[5] that includes elliptic curve cryptography arithmetic. C++ language is used for our implementation, along with fast in-line assembly language alternatives for most performance-critical parts of our code to speed up the performance, such as modular multiplication and exponentiation.

In order to evaluate the performance of the key generation of the OTS-SKE and KIS-based RA protocols that is performed on the proposed secure co-processor, we have additionally used with a slower in-order ARM architecture simulation with 64KB L1-I and L1-D cache, L1-I associativity of 2 and L1-D associativity of 4 using gem5 and compared its performance against the Intel key generation co-processor.

TABLE 1
Runtime Cost Analysis (in milliseconds) per Remote Attestation (RA) session of the bilinear map based OTS-SKE scheme in Comparison with the KIS-based RA and the ECDSA used in Intel SGX (Average over 100 runs). In the bilinear map based scheme we use $t = 4$ (such that $(t/\log_2 t) \cdot 64 = 128$ and $128/\log_2 t = 64$). The classical security of both schemes is 256 bits and a message signed during a RA session has 128 bits. The overhead of key generation that runs on the secure co-processor has been reported using ARM and Intel processors.

	OTS-SKE	KIS	ECDSA
Key Generation	1153.4 (ARM) 35.6 (Intel)	74.2 (ARM) 26.6 (Intel)	21.2
Signing	3.5	66.9	22.5
Verification	69.3	70.2	78.5

5.2 Runtime Cost Comparison

Table 1 shows the performance comparison of the KIS, OTS-SKE-based RA protocols with the standard Elliptic Curve Digital Signature Algorithm (ECDSA) based attestation used by Intel SGX. Both schemes are implemented with 256-bit security and the message signed during an RA session has 128 bits. The nonce used by the remote user has 64 bits. Note that the reported results correspond to a single session for OTS-SKE and KIS. For OTS-SKE, 64-bit nonce is represented by $n = 32$ t-ary symbols, where $t = 4$ ($64 = n \log_2 t$). This implies that OTS-SKE key generation produces $nt = 128$ subkeys in a single session. On the other hand, KIS produces a single key per session. ECDSA does not use sessions and pre-generates a single secret key, hence, we report the initial key generation time. The subkey generation of OTS-SKE has been parallelized using 32 threads on the Intel machine. However, on the in-order ARM machine, a serial key generation module has been used.

Key Generation. As it can seen from Table 1, with a serialized implementation of key generation in OTS-SKE (as used in the in-order ARM co-processor), the key generation cost of OTS-SKE (1.1 seconds) is significantly higher than the baselines, KIS and ECDSA (15× and 54× higher, respectively). This is because OTS-SKE generates 128 subkeys per session to create a unique secret signing key for each user. However, the generation of these subkeys can be significantly reduced by exploiting extreme parallelism. A parallel implementation on the faster Intel key generation co-processor using 32 threads has reduced this cost to 35.6 milliseconds (ms), which is comparable with the baselines.

Signing. In OTS-SKE, the signing process is completed by computing a string x based on the user's received random nonce and the message M and extracting a set of secret keys unique to x. For this reason, the generated unique secret session key can be directly forwarded to the user, rather than

4. Note that, in our scheme, even if signing goes wrong due to fault injections [25], [26], the security of our RA scheme is not broken. Because there are no secrets used in the signing procedure, injecting faults in the signing procedure cannot leak any extra information. The remote user will reject a wrong signature, so one just needs to sign again with a new session key from the co-processor. However, fault injections may hurt the correctness of results R produced by the AppEnc, therefore the AppEnc needs to use some form of fault-tolerant computing if fault injection attacks are present.

5. https://github.com/miracl/MIRACL

Algorithm 4 Bilinear-map based OTS-SKE

The implementation of the Key Generation and Signing modules of the OTP-SKE-EUF-CMA secure bilinear-map based OTS-SKE scheme. Algorithm IG generates a suitable mathematical structure for our signature scheme and λ is the security parameter. We assume a Pseudo Random Number Generator (PRNG) bootstrapped from an initial seed extracted from a True Random Number Generator (TRNG) to generate random bit strings.

1: **procedure** KEYGEN(session i)
2: $pk \leftarrow (p, \mathbb{G}, \mathbb{G}_1, e, g, g_1, g_2, h)$
3: $r_i \leftarrow$ PRNG
4: $aux_i \leftarrow g^{r_i}$
5: $l = g_2^{\alpha} h^{r_i}$
6: $k = g_1^{r_i}$
7: **for** $j \leftarrow 0$ to $n-1$ **do**
8: Generate V_j ▷ $\prod_{j=0}^{n-1} V_j = 1$
9: $m = lV_j$
10: **for** $b \leftarrow 0$ to $t-1$ **do**
11: $sk_{i,j,b} = m(k^{(it^n + bnt^j)})$
12: **end for**
13: **end for**
14: **end procedure**

1: **procedure** SIGN(session i, $nonce$, M)
2: Initialize $sk_{prod} \leftarrow 0$
3: $B = \text{PRP}(nonce; M)$ ▷ $B = \sum_{j=0}^{n-1} b_j t^j$
4: **for** $j \leftarrow 0$ to $n-1$ **do**
5: $sk_{prod} = sk_{prod} \cdot sk_{i,j,b_j}$
6: **end for**
7: $\sigma' \leftarrow (aux_i, sk_{prod})$
8: **return** $\sigma \leftarrow (\sigma', nonce)$
9: **end procedure**

performing an actual signing operation on message M. This reduces the computational overhead of OTS-SKE's signing (3.5 ms) since it only requires the selection and combination of a subset of keys. This allows OTS-SKE to avoid complex and costly elliptic curve operations during signing such as pairings. On the other hand, during the signing, KIS requires pairing on the elliptic curve which approximately takes 35 ms. This operation is repeated 3 times during signing. By parallelizing 2 pairings, we have reduced the signing cost of KIS from 105 ms to 66.9 ms, which is still significantly slower than the OTS-SKE. This can potentially become the main performance bottleneck in the runtime. Overall, OTS-SKE achieves 19× and 6.4× speedup over KIS and ECDSA, respectively.

Verification. This step is performed offline on the remote user side and therefore, its performance is not critical for the throughput of the RA protocol. However, as it can be seen from Table 1, the performance OTS-SKE, KIS and ECDSA-based RA protocols are comparable to each other during verification. As discussed in Section 4.1.2 During verification, pairing operation on the elliptic curve is performed 3 times. In our implementation, we have performed these 3 pairings in parallel, with takes 49.4 ms in total. Because of this, the overall cost of verification is high (70 ms on average for OTS-SKE and KIS), while the ECDSA takes 74.2 ms for verification. The performance of the verification process primarily depends on the overhead of the pairing operation (approximately 35 ms) being performed on the elliptic curve.

TABLE 2
Breakdown of Key Generation and Signing Modules of OTS-SKE given in Algorithm 4 (in milliseconds).

	Operation	Time	Repeats	Total Time
Key Generation	r_i	0.005	1	0.005
	aux_i	5.35	1	5.35
	l	7.34	1	7.34
	k	3.45	1	3.45
	V_j	0.11	32	3.52
	m	0.34	32	10.88
	$sk_{i,j,b}$	4.01	128	513.28
	Total			***543.8***
	Total (Parallel)			***35.5***
Signing	B	0.85	1	0.85
	sk_{prod}	0.083	32	2.65
	Total			***3.5***

5.3 Implementation and Runtime Cost Analysis of OTS-SKE based RA

Since the key generation and signing modules of the RA protocol are performed on the processor side, their performance is more critical compared to the verification module. For this reason, we provide the pseudocode for the implementation of the KEYGEN and SIGN modules of the bilinear-map based OTS-SKE in Algorithm 4, which have been introduced in Section 4.1.2, and show a breakdown of the runtime cost of each operation in Table 2. Note that, for the key generation, we only provide the breakdown of the secret key generation since the public key is generated once in the initialization phase, and hence, is not performed in run-time.

In KEYGEN, the runtime cost of the pre-generated variables r_i, aux_i, l and k to generate secret keys $sk_{i,j,b}$ are reported in Table 2, with a total runtime cost of 16 ms. V_j and m are generated $n = 32$ times and $sk_{i,j,b}$ is generated $nt = 128$ times, resulting in a total of 527 ms runtime cost without parallelization. In our implementation, we have parallelized the key generation loop (Line 7) given in Algorithm 4, and therefore, the total cost of the key generation has been reduced to 35.5 ms.

In SIGN, the computation of B to map the user's nonce with the message to a unique subset of the secret keys takes 0.85 ms in total. After this, $n = 32$ secret keys that belong to this subset are combined, i.e., multiplied, to create the final secret key sk_{prod}. Since this final key is created based on the user's nonce and message, it is directly returned by the SIGN module as the final signature. Therefore, the performance breakdown of the signature creation only includes the multiplication operation between $n = 32$ chosen secret subkeys, which takes 2.65 ms in total.

5.4 Performance Implications of Additional Hardware

The signing modules of both OTS-SKE and KIS run on the same enclave processor as the ECDSA-based RA protocol and the verification is offloaded to the remote user. Therefore, signing and verifications are performed in the same manner as the ECDSA-based RA and do not have additional performance implications. The main additional hardware

component required for the KIS and the OTS-SKE based RA is a secure key generation coprocessor, which we have evaluated in Table 1 using Intel and ARM processors. We have shown that, by duplicating the same Intel processor used by ECDSA as an additional co-processor for isolated key generation, and using a parallel implementation of the key generation module for OTS-SKE, we can make the computational cost of OTS-SKE comparable to the baseline KIS and ECDSA-based RA protocol.

The only additional requirement for OTS-SKE compared to the KIS is the use of a one-directional memory for key storage for a single session. While KIS only generates a single secret key for a session, OTS-SKE generates multiple subkeys (e.g., 128 given in Table 1). Therefore, OTS-SKE requires an additional 8 KB memory to store all subkeys, which is an inexpensive addition to the baseline KIS system.

5.5 Evaluation and Discussion

Today, the wide employment of elliptic curve cryptography (ECC) in various applications relies on a variety of implementation types from pure software or hardware implementations to hardware and software co-design. However, pure software implementations of ECC, despite offering the best flexibility at the lowest cost, cannot cope with the speed demands of many application areas as general purpose processors are not designed for efficient handling of ECC's underlying finite field arithmetic. Considering these limitations, hardware-based implementations turn out to be the more suitable alternatives [27], [28], [29], [30], [31], [32], [33]. Despite this, in this paper, we evaluate a software-based implementation of the ECC-based signature scheme that has its own computational disadvantages. However, we posit that with hardware acceleration and high parallelism, our proposed OTS-SKE RA scheme can achieve a significant performance boost.

Table 1 shows that the key generation phase is the main bottleneck in the OTS-SKE based signature scheme. The main contribution OTS-SKE is the use of one-time signatures for the RA in the secure processor architectures to protect the digital secrets against the adversary. This requires the processor to renew the secret keys, after each signing session, to be secure against impersonation attacks. This extra level of security increases the overhead of the key generation co-processor, which emerges as the main bottleneck in the OTS-SKE implementation. The introduction of a secure co-processor dedicated to key generation can potentially increase the overhead of the key generation due to the use of a slower processor. For example, on an in-order ARM processor, the key generation takes 1.1 seconds while its parallel implementation takes 35.6 ms on the main Intel processor. Even with the given slowdown with a slower co-processor, considering the fact that the key generation module runs in the background and in parallel with the signing, and remote attestation is done only once at the enclave (VM/container) creation time in state-of-the-art secure processors, this key generation cost is acceptable. For example, Kata container takes approx. 2.6 seconds to launch with AMD SEV [34].

The signing cost (3.4 ms) of the OTS-SKE based RA is much smaller than the baselines, while bringing better security benefits. Note that the introduction of a secure co-processor does not add additional overheads to the signing and verification phases. The verification takes place on the user side and the signing computations still take place in the remote attestation enclave on the main enclave processor. The only difference between the hardware implementations of the OTS-SKE's signing protocol and the baselines is the retrieval of the signing key. In the OTS-SKE based RA, the key needs to be retrieved from the special memory that consists of the current session keys. In our performance evaluation, we give an estimation based on key retrieval from DRAM. On the other hand, given the fact that the size of the entire set of session keys is 16KB, the proposed one-directional memory can be designed as a relatively small buffer with less latency than a DRAM. The signing cost of OTS-SKE is 19 and 6.4 times faster than the KIS and ECDSA, respectively. Therefore, even with multiple additional memory accesses, OTS-SKE remains faster during signing. Also, considering the fact that the remote attestation is performed only once at the enclave/VM/container creation time in state-of-the-art secure processors such as Intel SGX, AMD SEV, Intel TDX, the signing cost is relatively small. For example, Intel's EPID attestation takes 31.7 ms for quote generation and signing, at the enclave creation which takes 24.5 ms itself on an enclave with 5 MB of memory [35]. This shows that enclave creation itself already comes at a high cost. Additionally, the signing cost can further be reduced with pipelined hardware implementations of signing computations, B, and σ.

6 Conclusion

We demonstrated for the first time how to design a Remote Attestation (RA) protocol that resists a powerful adversary that can leak all digital secrets of a processor through side channels during the signing procedure. Even with secure processor technology that implements access control using hardware isolation but without any privacy guarantees (due to the recent avalanche of attacks), our remote attestation is secure and can be used to verify computation by remote users. The new RA scheme offers the first crucial level of trust for current attacked secure processor technology.

Acknowledgments

This research was supported by the National Science Foundation under Grants No. 1617774 and 1929261.

References

[1] V. Costan and S. Devadas, "Intel sgx explained," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 86, 2016.
[2] Intel, "Intel trust domain extensions," 2020. [Online]. Available: https://software.intel.com/content/dam/develop/external/us/en/documents/tdxwhitepaper-v4.pdf
[3] D. Kaplan, J. Powell, and T. Woller, "Amd memory encryption whitepaper," 2016.
[4] Arm, "Arm confidential compute architecture (arm cca)," 2021. [Online]. Available: https://developer.arm.com/architectures/architecture-security-features/confidential-computing
[5] ARM, "Arm security technology – building a secure system using trustzone technology." *ARM Technical White Paper*, 2009.
[6] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," in *SIGP*, 2000.
[7] G. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "Aegis: architecture for tamper-evident and tamper-resistant processing," in *ICS*, 2003.

[8] D. Champagne and R. Lee, "Scalable architectural support for trusted software," *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, pp. 1–12, 2010.

[9] R. Boivie and P. Williams, "Secureblue++: Cpu support for secure execution," *IBM, IBM Research Division, RC25287 (WAT1205-070)*, pp. 1–9, 2012.

[10] C. W. Fletcher, M. van Dijk, and S. Devadas, "A secure processor architecture for encrypted computation on untrusted programs," in *STC '12*, 2012.

[11] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *USENIX Security Symposium*, 2016.

[12] T. Bourgeat, I. Lebedev, A. Wright, S. Zhang, Arvind, and S. Devadas, "Mi6: Secure enclaves in a speculative out-of-order processor," *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.

[13] H. Omar and O. Khan, "IRONHIDE: A secure multicore that efficiently mitigates microarchitecture state attacks for interactive applications," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22-26, 2020*. IEEE, 2020, pp. 111–122. [Online]. Available: https://doi.org/10.1109/HPCA47549.2020.00019

[14] A. Nilsson, P. N. Bideh, and J. Brorsson, "A survey of published attacks on intel sgx," *ArXiv*, vol. abs/2006.13598, 2020.

[15] S. Checkoway and H. Shacham, "Iago attacks: Why the system call API is a bad untrusted RPC interface," in *Proceedings of ASPLOS 2013*, R. Bodik, Ed. ACM Press, Mar. 2013, pp. 253–64.

[16] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *Science*, vol. 283, pp. 1237 – 1237, 1999.

[17] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1–19, 2019.

[18] F. Dall, G. D. Micheli, T. Eisenbarth, D. Genkin, N. Heninger, A. Moghimi, and Y. Yarom, "Cachequote: Efficiently recovering long-term secrets of sgx epid via cache attacks," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, pp. 171–191, 2018.

[19] R. Buhren, C. Werling, and J.-P. Seifert, "Insecure until proven updated: Analyzing amd sev's remote attestation," *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[20] Y. Dodis, J. Katz, S. Xu, and M. Yung, "Key-Insulated Public Key Cryptosystems," in *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, 2002, pp. 65–82.

[21] ——, "Strong Key-Insulated Signature Schemes," in *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, 2003, pp. 130–144.

[22] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "One-time programs," in *Annual International Cryptology Conference*. Springer, 2008, pp. 39–56.

[23] J. Kilian, "Founding crytpography on oblivious transfer," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, 1988, pp. 20–31.

[24] S. S. Chow, L. C. Hui, S. M. Yiu, and K. Chow, "Secure hierarchical identity based signature and its application," in *International Conference on Information and Communications Security*. Springer, 2004, pp. 480–494.

[25] K. Murdock, D. Oswald, F. Garcia, J. V. Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against intel sgx," *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1466–1482, 2020.

[26] Z. Kenjar, T. Frassetto, D. Gens, M. Franz, and A.-R. Sadeghi, "V0ltpwn: Attacking x86 processor integrity from software," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 1445–1461.

[27] G. Agnew, R. Mullin, and S. Vanstone, "An implementation of elliptic curve cryptosystems over f2155," *IEEE J. Sel. Areas Commun.*, vol. 11, pp. 804–813, 1993.

[28] K. H. Leung, K. W. Ma, W. Wong, and P. Leong, "Fpga implementation of a microcoded elliptic curve cryptographic processor," *Proceedings 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (Cat. No.PR00871)*, pp. 68–76, 2000.

[29] L. Gao, S. Shrivastava, and G. Sobelman, "Elliptic curve scalar multiplier design using fpgas," in *CHES*, 1999.

[30] Y. Zhang, C. Xue, D. Wong, N. Mamoulis, and S. Yiu, "Acceleration of composite order bilinear pairing on graphics hardware," *IACR Cryptol. ePrint Arch.*, vol. 2011, p. 196, 2012.

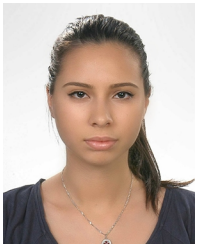
[31] S. Cui, J. Großschädl, Z. Liu, and Q. Xu, "High-speed elliptic curve cryptography on the nvidia gt200 graphics processing unit," in *ISPEC*, 2014.

[32] W. Pan, F. Zheng, Y. Zhao, W. Zhu, and J. Jing, "An efficient elliptic curve cryptography signature server with gpu acceleration," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 1, pp. 111–122, 2017.

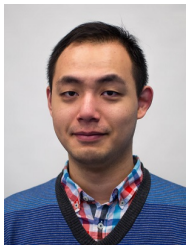
[33] S. Pu and J. Liu, "Eagl: An elliptic curve arithmetic gpu-based library for bilinear pairing," in *Pairing*, 2013.

[34] J. Gu, X. Wu, B. Zhu, Y. Xia, B. Zang, H. Guan, and H. Chen, "Enclavisor: A hardware-software co-design for enclaves on untrusted cloud," *IEEE Transactions on Computers*, vol. 70, no. 10, pp. 1598–1611, 2021.

[35] K. A. Küçük, A. Paverd, A. Martin, N. Asokan, A. Simpson, and R. Ankele, "Exploring the use of intel sgx for secure many-party applications," in *Proceedings of the 1st Workshop on System Software for Trusted Execution*, ser. SysTEX '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/3007788.3007793



Deniz Gurevin received her B.S. degree in Electrical and Electronics Engineering from MEF University, Istanbul, Turkey in 2018. She is currently pursuing her Ph.D. degree and working as a graduate research assistant in the Department of Electrical & Computer Engineering at the University of Connecticut, CT, USA. Her main research interests include hardware security, spatiotemporal graph processing and deep learning.



Chenglu Jin received the Ph.D. degree from the Electrical and Computer Engineering Department, University of Connecticut, Storrs, CT, USA, in 2019. He is a tenure-track researcher in the Computer Security Group in Centrum Wiskunde & Informatica (CWI Amsterdam), The Netherlands. His research interests are cyber–physical system security, hardware security, and applied cryptography.

Phuong Ha Nguyen received the Specialist degree in computer science and mathematics from Lomonosov Moscow State University, Russia in 2008, and the Ph.D. degree in cryptography from Nangyang Technological University, Singapore, in 2013. He is currently working as a Researcher at eBay. His research interests include machine learning and cryptography.

Omer Khan is the Castleman Associate Professor in the Department of Electrical & Computer Engineering at the University of Connecticut. Prior to joining UConn, he was a Postdoctoral Research Scientist at the Massachusetts Institute of Technology. His research interests include developing cross-layer methods to improve the performance scalability and security of multicore processor architectures. Khan received a PhD in Electrical and Computer Engineering from the University of Massachusetts Amherst. He is a senior member of IEEE and a member of ACM.

Marten van Dijk (Fellow, IEEE) is currently a Group Leader of the Computer Security Group, CWI, The Netherlands, with over 20 years of experience in both industry (Philips Research and RSA Laboratories) and academia (MIT and UConn). His work has been recognized by the A. Richard Newton Technical Impact Award in electronic design automation, in 2015, and has received several best (student) paper awards.