Real-Time Digital Signatures for Time-Critical Networks

Attila A. Yavuz Member, IEEE, Anand Mudgerikar, Ankush Singla, Ioannis Papapanagiotou Senior Member, IEEE, and Elisa Bertino Fellow, IEEE

Abstract—The secure and efficient operation of time-critical networks, such as vehicular networks, smart-grid and other smart-infrastructures, is of primary importance in today’s society. It is crucial to minimize the impact of security mechanisms over such networks so that the safe and reliable operations of time-critical systems are not being interfered. For instance, if the delay introduced by the crypto operations negatively affects the time available for braking a car before a collision, the car may not be able to safely stop in time. In particular, as a primary authentication mechanism, existing digital signatures introduce a significant computation and communication overhead, and therefore are unable to fully meet the real-time processing requirements of such time-critical networks.

In this paper, we introduce a new suite of real-time digital signatures referred to as Structure-free and Compact Real-time Authentication (SCRA), supported by hardware acceleration, to provide delay-aware authentication in time-critical networks. SCRA is a novel signature framework that can transform any secure aggregate signature into a signer efficient signature. We instantiate SCRA framework with Condensed-RSA, BGLS, and NTRU signatures. Our analytical and experimental evaluation validates the significant performance advantages of SCRA schemes over their base signatures and the state-of-the-art schemes. Moreover, we push the performance of SCRA schemes to the edge via highly optimized implementations on vehicular capable System-on-Chip (SoC) as well as server-grade General Purpose Graphics Processing Units (GPGPUs). We prove that SCRA is secure (in random oracle model) and show that SCRA can offer an ideal alternative for authentication in time-critical applications.

Index Terms—Applied cryptography; digital signatures; real-time authentication; hardware-acceleration.

I. INTRODUCTION

Technological advances in sensors and embedded systems are making the deployment of "smart" infrastructures possible. Such infrastructures will usher automation in a large number of application domains such as transportation, manufacturing, smart-grid and urban life (e.g., Smart-city).

Because of their control capabilities and pervasive data acquisition, securing such smart-infrastructures is a critical requirement. Even though many security techniques are available, their application to smart infrastructures is not straightforward, especially when such infrastructures are based on networks that include mobile devices, and for safety reasons, they have to meet real-time requirements. We refer to such networks as time-critical networks.

An example is a vehicular network in which events from vehicles, such as sudden brake of a vehicle, have to be communicated promptly to the other vehicles in the network so that they can timely react to the events. Scalability is also crucial as many envisioned time-critical networks involve huge numbers of devices and systems. A key security technique for any comprehensive solution is represented by authentication as it is critical for establishing trust and securing communications among parties in a network. Authentication techniques have been widely investigated. However, to meet the real-time and scalability requirements of large scale time-critical networks, we need techniques that are far more efficient than the currently available ones. It is critical that devices in such a network should be able to respond and/or to initiate a large number of authentications in a small time-frame.

To address such a requirement, in this paper we develop a series of fast digital signatures, supported by hardware-acceleration, to enable real-time authentication in time-critical networks. We introduce a generic signature framework, referred to as Structure-free and Compact Real-time Authentication (SCRA), that can be instantiated with any secure aggregate signature. We then develop specific SCRA instantiations from Condensed-RSA [30], BGLS [7], NTRU [27] and PASSSign [18], and demonstrate that these SCRA schemes are significantly more computationally efficient than their counterparts in modern CPUs. We also computationally parallelize SCRA across thousands lightweight threads commonly supported by modern GPUs. We use several optimizations and show that the performance can be higher compared to the performance obtained when the CPU is used. Finally, we apply similar optimizations to SoCs commonly used by car manufacturers and IoT deployments.

A. State-of-the-Art Methods and Limitations

We outline the advantages and limitations of authentication mechanisms that are most relevant to our work.
Message Authentication Codes and Standard Digital Signatures: Symmetric crypto-based authentication mechanisms rely on Message Authentication Code (MAC) [28]. Despite their computational efficiency, these methods are not practical for broadcast authentication in large-scale distributed systems, as they require pairwise key distribution among all signers and verifiers. They also cannot achieve non-repudiation and public verifiability. Digital signatures (e.g., RSA [35], ECDSA [3]) rely on the Public Key Infrastructures (PKIs) [28], which makes them publicly verifiable and scalable for large systems. Hence, they are considered as a primary authentication mechanism for large-scale delay-aware systems. For instance, the vehicular WAVE architecture mandates the use of PKI mechanisms to sign critical messages [2]. Despite their scalability, standard digital signature schemes require several expensive operations such as modular exponentiation and pairing (e.g., BLS [8]). Therefore, they are not suitable for time-critical authentication. It has been shown that they introduce significant delays, which are unacceptable in time-critical networks such as vehicular networks [33].

Delayed Key Disclosure and Amortized Signatures: Delayed key disclosure methods [32] are efficient as they introduce an asymmetry between signer and verifier via a time factor. However, these methods require packet buffering, and therefore cannot achieve immediate verification (which is vital for delay-aware authentication). Signature amortization (e.g., [25]) computes a signature over a set of messages instead of individual messages. Hence, the cost of signature generation and verification is amortized over multiple messages. However, these methods require packet buffering and introduce packet loss risk due to the use of hash chains.

Specialized Signatures: One-Time Signatures (OTSs) (e.g., [34]) offer fast signature generation and verification. However, they incur very large signature and public key sizes, and also public keys must be renewed frequently. Various customizations of traditional signatures (along with cryptographic pairing [8]) and OTSs for time-critical systems such as vehicular networks (e.g., [16]) and smart-grids have been proposed. However, these schemes still suffer from computational inefficiency (due to heavy use of pairings) or public key distribution issues (OTSs).

The offline-online signatures (e.g., [31]) pre-compute a token for each message to be signed at the offline-phase, and then use it to compute a signature on a message very efficiently at the online-phase. Despite their merits, offline-online signatures incur significant storage overhead (i.e., linear with respect to the number of messages to be signed). Moreover, they require heavy computation for applications with high message throughput, since the signer depletes pre-computed tokens rapidly and is forced to regenerate them at the online-phase. Hence, offline-online signatures are not suitable for time-critical networks with high message throughput.

Our prior work Rapid Authentication (RA) [41] is an efficient offline-online signature, which leverages the already
available pre-defined message structures in certain applications (e.g., smart-grid) to reduce the computational and storage overhead of RSA-type offline-online constructions. Despite its advantages, RA is only suitable for applications that have a pre-defined message structure with a limited number of message components. Moreover, RA requires pre-computed tokens (i.e., one-time masking signatures) to be stored/renewed per item as in traditional offline-online techniques. Hardware-Accelerated Authentication (HAA) [39] exploits hardware acceleration to speed up RA in various settings. HAA demonstrates the benefit of hardware acceleration to reduce the end-to-end delay of digital signature schemes. In particular, HAA shows the performance advantages offered by GPUs for offline-online signatures to batch regenerate tokens as they are depleted.

B. Our Contribution
We develop a new suite of delay-aware signatures that we refer to as Structure-Free and Compact Authentication (SCRA) to enable fast authentication for time-critical networks.

Main Idea: SCRA is based on the observation that the signature aggregation operation of some signature schemes is several magnitudes of times faster than that of their signature generation. We leverage this fact to shift the expensive operations of signature generation phase to the key generation phase. That is, at the key generation (offline), we compute a set of signatures on the bit-structures of a hash output domain. Later, we can combine these pre-computed signatures very efficiently based on the hash of each message without enforcing a message format (e.g., unlike [41]) or storage/regeneration of a token per-message (e.g., unlike offline-online signatures (e.g., RA [41], [9]) that incurs linear storage and re-computation overhead). This simple but elegant strategy enables SCRA to achieve very fast signature generation, a low end-to-end cryptographic delay, small-constant signature sizes with a constant-size private/public key. Figure 1 further outlines our main idea.

Properties: We outline below the relevant properties of our schemes.
- **Generic and Simple Design:** SCRA can be instantiated from any aggregate signature. We prove that SCRA is EU-CMA-secure if its base scheme is IA-EU-CMA secure (see Section II). We show that SCRA is at least a magnitude times faster than standard signatures as shown in Table I even without optimization.
- **Highly Fast Signing, Low Delay and Compactness:** We develop several instantiations of SCRA offering performance trade-offs with different computational overhead, signature and key sizes.
  - SCRA-BGLS is constructed from BGLS [7], which reduces the signing cost from an exponentiation to a few modular multiplications. SCRA-BGLS offers the smallest signature size among all counterparts with a minimal signer overhead, making it suitable for resource-limited devices.
  - SCRA-NTRU is based on the NTRU [27] signature scheme. It is important to mention that we use the NTRU scheme that is secure against transcript attacks [13]. Signatures are aggregated using the lattice based aggregation technique described in [15]. The lattice based sequential aggregate signature is proven to be secure in the random oracle security model [4]. Due to its moderate signature and key sizes and low end-to-end delay, SCRA-NTRU is ideal for time-critical applications.
  - SCRA-NTRUPASS is based on the PASS [18] signature scheme. It is also a lattice based cryptographic scheme based on the partial Fourier recovery problem.
- **Performance Enhancements via Hardware-Acceleration:** We improve the performance of SCRA by developing various hardware-acceleration and software-optimizations, which enable significant speed improvements (see Section VI).

II. DEFINITIONS AND MODELS
We first introduce our notation and definitions, followed by our system and threat model. We then give our security model, in which we clarify the security properties of the SCRA schemes.

A. Notation and Definition

\(|S|\) denotes the cardinality of set \(S\). \(\{x_i\}_{i=0}^l\) denotes \((x_0, \ldots, x_l)\). \(x \sim S\) denotes that variable \(x\) is randomly and uniformly selected from set \(S\). ||, |\(x|\) and \(\{0, 1\}^*\) denote the concatenation operation, the bit length of variable \(x\) and the set of binary strings of any finite length, respectively.

**Definition 1** A signature scheme \(\text{SGN}\) is a tuple of three algorithms \((\text{Kg}, \text{Sig}, \text{Ver})\) defined as follows:
- \((sk, PK) \leftarrow \text{SGN}.\text{Kg}(1^n)\): Given the security parameter \(1^n\), the key generation algorithm returns a private/public key pair \((sk, PK)\) as the output.
- \(s \leftarrow \text{SGN}.\text{Sig}(m, sk)\): The signing algorithm takes a message \(m \in \{0, 1\}^*\) and a private key \(sk\) as the input, and returns a signature \(s\) as the output.
- \(\{0, 1\} \leftarrow \text{SGN}.\text{Ver}(m, s, PK)\): The verification algorithm takes a message \(m \in \{0, 1\}^*\), signature \(\sigma\) and public key \(PK\) as the input. It returns a bit: 1 means valid and 0 means invalid.
\[(sk, PK) \leftarrow A\text{Sig}.Kg(1^*)\): Given the security parameter \(1^*\), the key generation algorithm returns a private/public key pair \((sk, PK)\) as the output.

- \(\gamma_i \leftarrow A\text{Sig}.\text{Sign}(m_i, sk):\) The signing algorithm takes a message \(m_i \in \{0,1\}^*\) and private key \(sk\) as the input. It returns a signature \(\gamma_i\) computed under \(sk\) as the output.

- \(s \leftarrow A\text{Sig}.\text{Agg}(\gamma_1, \ldots, \gamma_L, \text{params}):\) The aggregation algorithm takes a set of signatures \(\gamma_1, \ldots, \gamma_L\) and optionally some parameters \(s\) as the input. It returns a single-compact signature \(s\) as the output. Optional parameters may include \(s\) (aggregation under private key) or \(PK\) (public aggregation) depending on specific instantiations. We will omit \(s\) for the sake of simplicity.

- \(\{0,1\} \leftarrow A\text{Sig}.\text{Ver}(\overline{m}, s, PK):\) The verification algorithm takes messages \(\overline{m} = (m_1, \ldots, m_L)\), aggregate signature \(s\) and \(PK\) as the input. It returns a bit: \(1\) means valid and \(0\) means invalid.

### B. System and Threat Model

Our system model follows the traditional PKC-based broadcast authentication model (e.g., [41]), in which a signer computes a digital signature on a message and broadcasts a message-signature pair to the verifiers. This model is compatible with our target time-critical applications. For instance, in vehicular networks, a vehicle or road infrastructure broadcasts authenticated messages to the surrounding entities as described in vehicular communication standards [2]. Our threat model reflects how a standard digital signature-based broadcast authentication works. That is, an adversary \(A\) can observe message-signature pairs computed under a private key. \(A\) also can actively intercept, modify, inject and replay messages transmitted over the network. \(A\) aims at producing existential forgeries against the digital signatures computed by signers.

### C. Security Model

The security notion for a signature is **Existential Unforgeability under Chosen Message Attacks (EU-CMA).**

**Definition 3** The EU-CMA experiment for \(SGN\) is as follows:

- **Setup.** Algorithm \(B\) runs \((sk, PK) \leftarrow SGN.Kg(1^*)\) and provides \(PK\) to the adversary \(A\).
- **Queries.** \(A\) queries \(B\) on any message \(m_j\) of her choice for \(j = 1, \ldots, q_s\). \(B\) replies to each query with a signature \(s_j \leftarrow SGN.\text{Sign}(m_j, sk)\).
- **Forgery.** \(A\) outputs a forgery \((m^*, s^*)\) and wins the EU-CMA experiment, if \(SGN.\text{Ver}(PK, m^*, s^*) = 1\) and \(m^*\) was not queried to \(B\).

\(SGN\) is \((t, q_s, \epsilon)\)-EU-CMA secure, if no \(A\) in time \(t\) making at most \(q_s\) queries has an advantage with probability \(\epsilon\).

**SCRA** is constructed from a single-signer aggregate signature that achieves the signature immutability (described in detail below). The basic security notion for aggregate signatures is **Aggregate-EU-CMA (A-EU-CMA)** [20], [7], which captures the homomorphic properties of aggregate signatures. Later, the security of aggregate signatures has evolved to capture improved security properties such as **signature immutability**. Intuitively, signature immutability refers to the difficulty of computing new valid aggregated signatures from a set of other aggregated signatures [29]. To describe **Immutable-A-EUCMA (IA-EU-CMA)** [26], [43] security, we first define the aggregate signature extraction argument as below.

**Aggregate Signature Extraction:** The \(L\)-aggregate signature extraction problem, referred as \(AE\) problem, means that for a given aggregate signature \(s \leftarrow A\text{Sig}.\text{Agg}(\gamma_1, \ldots, \gamma_L)\) computed on \(L\) individual data items, it is difficult to extract the individual signatures \((\gamma_1, \ldots, \gamma_L)\) provided that only \(s\) is known to the extractor. Moreover, it is difficult to extract any aggregate signature subset \(s'\) from a given aggregate signature \(s\) [42]. The \(AE\) problem was first introduced by Boneh et al. in [7] for the security of BGLS signatures, but as an intractability assumption without a proof. Coron et al. in [10] later showed that Boneh’s \(AE\) problem for BGLS scheme is equivalent to the Computational Diffie Hellman Assumption (CDH) [21]. Yavuz et al. in [43] analyzed the log truncation problem for forward-secure and aggregate signatures [26], and produced formal proofs with \(AE\) argument for only the DLP-based schemes [43]. A related problem in the context of one-way accumulators for RSA have been considered in [6], which extends to other aggregate RSA variants (e.g., C-RSA [29]).

**Definition 4** The \(AE\) experiment for a \(A\) is as follows [42]:

- **Setup.** Algorithm \(B\) runs \((sk, PK) \leftarrow A\text{Sig}.Kg(1^*)\) and provides \(PK\) to the adversary \(A\).
- **Queries.** \(A\) queries \(B\) on any batch message comprised of \(L\) individual messages \(\overline{m}_j = (m_{j,1}, \ldots, m_{j,L})\) of her choice for \(j = 1, \ldots, q_b\). \(B\) replies to each query \(j\) with an aggregate signature \(s_j \leftarrow A\text{Sig}.\text{Agg}(\gamma_{j,1}, \ldots, \gamma_{j,L})\), where \(\gamma_{j,i} \leftarrow A\text{Sig}.\text{Sign}(m_{j,i}, sk)\) \((1 \leq i \leq L)\).
- **Aggregate Extraction.** \(A\) outputs \((\overline{m}^*, \sigma')\), where \(\overline{m}^* = (m_{1,k}^*, \ldots, m_{k,L}^*)\), \(1 \leq k \leq L\) and wins the \(AE\) experiment, if

1. \(A\text{Sig}.\text{Ver}(\{m_{i}^*\}_{i \in \{1, \ldots, k\}}, \sigma', PK) = 1\),
2. \(\overline{m}^*\) is a subset of previously queried or some combination of previously queried batch messages: \(\exists \overline{m'} \subseteq \{1, \ldots, q_b\} : \overline{m}^* \subseteq \{\overline{m}_i\}_{i \in \overline{m'}}\). This implies that \(\overline{m}^*\) itself as a batch query never has been queried directly to \(B\) (but individual data items in \(\overline{m}^*\) have been queried as an element of different batch queries before, but not individually),
3. The extraction is non-trivial: If \(\overline{m}^*\) is combined with any previously queried or a combination of previously queried batch messages, the combination is not equal to one of the previously queried batch message itself: \(\forall \overline{m} \subseteq \{1, \ldots, q_b\} : (\overline{m}^* \cap \{\overline{m}_i\}_{i \in \overline{m}}) \neq \{\overline{m}_i\}_{i \in \overline{m}}\).

\(A\text{Sig}\) is \((t, q_b, \epsilon)\)-\(AE\) secure, if no \(A\) in time \(t\) making at most \(q_b\) queries has an advantage with probability \(\epsilon\).

We now provide the definition of **Immutable-A-EUCMA (IA-EU-CMA)** security [26], [43] as below:
Definition 5: The IA-EU-CMA experiment for a $\text{ASig}$ is as follows [42]:

- Setup. Algorithm $B$ runs $(sk, PK) \leftarrow \text{ASig.Kg}(1^n)$ and provides $PK$ to the adversary $A$.

- Queries. $A$ queries $B$ on any batch message comprised of $L$ individual messages $\overrightarrow{m}_j = (m_{j,1}, \ldots, m_{j,L})$ of her choice for $j = 1, \ldots, q$. $B$ replies to each query $j$ with an aggregate signature $s_j \leftarrow \text{ASig.Aggs}(\gamma_{j,1}, \ldots, \gamma_{j,L})$, where $\{\gamma_{j,i} \leftarrow \text{ASig.Sig}(m_{j,i}, sk)\}_{i=1}^L$.

- Forgery. $A$ outputs a forgery $(\overrightarrow{m}^*, \gamma^*)$ and wins the experiment IA-EU-CMA, if

1. The forgery is valid as $\text{ASig.Ver}(\overrightarrow{m}^*, \gamma^*, PK) = 1$,
2. $\overrightarrow{m}^*$ is a subset of previously queried or some combination of previously queried batch messages: $\exists I' \subseteq \{1, \ldots, q\} : \overrightarrow{m}^* \subseteq \big||_{k \in I'} \overrightarrow{m}_k$,
3. Batch query $\overrightarrow{m}^*$ has not been queried previously as $\overrightarrow{m}^* \not\subseteq \big\{ \overrightarrow{m}_j \big\}_{j=1}^{j=1}$.

$\text{ASig}$ is $(t, q, \epsilon)$-IA-EU-CMA-secure, if no $A$ in time $t$ making at most $q$ queries has an advantage at least with probability $\epsilon$.

III. PROPOSED SCHEMES

In this section, we present our proposed schemes. We first describe the SCRA-Adaptive digital signature framework. We then provide several instantiations of the generic SCRA, each offering a unique performance benefit compared to the others.

A. Structure-free and Compact Real-time Authentication

SCRA can transform any aggregate signature into a signer-efficient signature scheme, whose signing operation is as fast as just the aggregation (i.e., simple modular addition or multiplication) of a small set of pre-computed signatures. SCRA has several advantages over the state-of-the-art signature schemes: (i) SCRA is a magnitude(s) of times more efficient with respect to signature generation than standard signatures (e.g., RSA [35], ECDSA [3], BGLS [7]). (ii) Unlike message-formatted signature schemes [41], SCRA does not require any pre-defined message formats. (iii) Unlike offline-online signatures [37], [31], [41], SCRA does not require linear-sized token storage. (iv) SCRA offers compact signature and public key sizes, and therefore is more scalable than one-time signatures (e.g., [34]).

The detailed description of SCRA is given in Algorithm 1.

We further elaborate as follows:

Let $(sk', PK') \leftarrow \text{ASig.Kg}(1^n)$ be a $\text{ASig}$ key pair and $H : \{0,1\}^* \rightarrow \{0,1\}^d$ be an ideal hash function (i.e., $H$ behaves as a Random Oracle (RO) [4]), where $d$-bit denotes the output length of the cryptographic hash function.

1) Key Generation (Offline): We apply a divide-and-conquer strategy over the hash output $H : \{0,1\}^* \rightarrow \{0,1\}^d$. That is, a $d$-bit hash output can be interpreted as integers $(j_1, \ldots, j_d)$, where each $j_i$ is a $b$-bit integer such that $b \cdot L = d$. We then compute a signature on each $b$-bit integer $j$ with its corresponding index $i$ as $\tilde{m}_{i,j} \leftarrow i||j||P^i, \gamma_{i,j} \leftarrow \text{ASig.Sig}(\tilde{m}_{i,j}, sk')$, $i = 1, \ldots, L$, $j = 0, \ldots, 2^b - 1$, where $P$ is a random padding. The index $(i, j)$ will enable the signer to select the corresponding pre-computed signature from the table $\Gamma$ in the online phase for a given message, and therefore ensure the correctness of the scheme. Moreover, the index $i$ enforces the order of the bit chunks in the online phase. The random padding $P$ is added to ensure that, for practical applications, the input of hash function remains larger than $d$ as required.

We construct a pre-computed sub-message/signature table $\Gamma = \{\tilde{m}_{i,j}, \gamma_{i,j}\}_{i=1, j=0}^{L, 2^b-1}$ which supports very efficient signature generation. $\Gamma$ is constant-size (e.g., unlike [31], [9]) and imposes no structure/length constraints on the online messages to be signed (e.g., unlike [41]).

2) Signature Generation: Given $m \in \{0,1\}^*$, the signer computes $(M_1^*, \ldots, M_L^*) \leftarrow H(m||r)$, and fetches the corresponding signatures $\gamma_i^*$ of $i||M_i^*||P$ from the table, where $\gamma_i^* \in \{0,1\}^*$. $i = 1, \ldots, L$. The rest is to combine signatures efficiently as $s \leftarrow \text{ASig.Aggs}(\gamma_1^*, \ldots, \gamma_L^*)$, where $s \leftarrow (r, s)$.

3) Signature Verification: The verifier computes $(M_1^*, \ldots, M_L^*) \leftarrow H(m||r)$ and verifies $s$ as $s \leftarrow (0,1) \leftarrow \text{ASig.Ver}(1||M_1^*||P, \ldots, L||M_L^*||P, s, PK')$.

B. Instantiations of SCRA

An ideal aggregate signature to instantiate SCRA must achieve very efficient signature aggregation and IA-EU-CMA security. We identified three signatures to instantiate SCRA: Condensed-RSA (C-RSA) [30] based on RSA [35], BGLS [7] based on pairing and aggregate-NTRU signatures [36], [15] based on NTRU [13]. We summarize important operations of our SCRA instantiations in Algorithms 2-5. For the sake of brevity, we only give the dominant signature operations that are performed in each algorithm. The rest of the SCRA operations are as described in Algorithm 1 and are not repeated. Moreover, we only give the private/public keys of each instantiation without describing key generation steps and parameters in detail. We refer interested readers to C-RSA [30], BGLS [7] and NTRU [36], [15] for the details.

SCRA-C-RSA is based on Condensed-RSA (C-RSA) [30] and therefore it obtains the highest computational efficiency benefit from SCRA among all instantiations. That is, C-RSA is by default a verifier efficient signature scheme but its signature generation is expensive (i.e., an exponentiation under a large modulo). Since the SCRA significantly reduces the signing cost, SCRA-C-RSA achieves the lowest end-to-end delay among all instantiations with a moderate signature size (e.g., 2KB RSA signature size). SCRA-C-RSA is described in Algorithm 2.

SCRA-BGLS is based the BGLS signatures [7], and therefore has the smallest signature/key size among all instantiations (e.g., 320 bits). The SCRA strategy also significantly increases the signature efficiency of BGLS. However, since BGLS has an expensive signature verification due to cryptographic pairing operations, SCRA-BGLS has a larger end-to-end cryptographic delay compared to our other instantiations. SCRA-BGLS is described in Algorithm 3.
Algorithm 1 Structure-free Compact Real-Time Authentication (SCRA) Scheme

\[(sk, PK) \leftarrow \text{SCRA.Agg}(1^k)\]: Executed offline (once).
1. \((sk', PK') \leftarrow \text{ASig.Kg}(1^k), P \overset{\$}{\leftarrow} \{0,1\}^d.\)
2. Select integers \((b, L)\) such that \(b \cdot L = d.\)
3. \(m_{i,j} \leftarrow [j]_P, \gamma_{i,j} \leftarrow \text{ASig.Sig}(m_{i,j}, sk^k), i = 1, \ldots, L, j = 0, \ldots, 2^b - 1.\)
4. \(sk \leftarrow (sk^k, \Gamma)\) and \(PK \leftarrow (PK^k, P), \) where \(\Gamma \leftarrow \{m_{i,j}, \gamma_{i,j}\}\) for \(i = 1, \ldots, L, j = 0, \ldots, 2^b - 1.\)

\(\sigma \leftarrow \text{SCRA.Sig}(m, sk)\): Given a message \(m \in \{0,1\}^*,\) compute its signature as follows:
1. \((M_1^i, \ldots, M_L^i) \leftarrow H(m||r),\) where \(r \leftarrow \{0,1\}^*\) and \(M_i^i \in \{0,2^b - 1\}, i = 1, \ldots, L.\)
2. \(m'_i \leftarrow i||M_i^i||P, i = 1, \ldots, L,\)
3. \(s \leftarrow \text{ASig.Agg}(\gamma_{1,1}, \ldots, \gamma_L, \sigma)\) and \(\sigma = (r, s).\)

\(\{0,1\} \leftarrow \text{SCRA.Ver}(m, \sigma, PK)\): Given \(m \in \{0,1\}^*\), verify its signature \(\sigma\) under \(PK\) as follows:
1. \((M_1^i, \ldots, M_L^i) \leftarrow H(m||r),\)
2. \(m'_i \leftarrow i||M_i^i||P, i = 1, \ldots, L,\)
3. \(\{0,1\} \leftarrow \text{ASig.Ver}(m_1', \ldots, m_L'), s, PK').\)

Algorithm 2 SCRA instantiation with Condensed-RSA [30]: SCRA-C-RSA

\((sk, PK) \leftarrow \text{SCRA-C-RSA.Kg}(1^k)\): Given \(1^k,\) generate C-RSA and SCRA-C-RSA parameters as follows:
1. Randomly generate two large primes \((p, q)\) and computes \(n = p \cdot q.\) The public and secret exponents \((e, d) \in \mathbb{Z}_n^*\) satisfies \(e \cdot d \equiv 1 \mod \phi(n),\) where \(\phi(n) = (p - 1)(q - 1).\) Set \(sk^i \leftarrow (n, d)\) and \(PK^i \leftarrow (n, e).\) Let \(H'\) be a full domain hash function (e.g., [5]) defined as \(H' : \{0,1\}^* \to \mathbb{Z}_n.\)
2. Compute \(\gamma_{i,j} \leftarrow H'(m_{i,j})^d \mod n, i = 1, \ldots, L, j = 0, \ldots, 2^b - 1,\) set \((\Gamma, sk, PK)\) as in Algorithm 1.

\(\sigma \leftarrow \text{SCRA-C-RSA.Sig}(m, sk)\): Execute Algorithm 1 SCRA.Sig Step 1-2, and obtain \(\gamma_i\) of \(m_i^i\) from \(\Gamma, i = 1, \ldots, L.\) Compute \(s \leftarrow \prod_{i=1}^L (\gamma_i) \mod n.\) Set \(\sigma\) as in Algorithm 1 SCRA.Sig Step 3.

\(\{0,1\} \leftarrow \text{SCRA-C-RSA.Ver}(m, \sigma, PK)\): Execute Algorithm 1 SCRA.Ver Step 1-2. In Step 3 (i.e., aggregate signature verification), if \(s^e = \prod_{i=1}^L H'(m_i^i) \mod n, return 1,\) else return 0.

Algorithm 3 SCRA instantiation with BGLS [7]: SCRA-BGLS

\((sk, PK) \leftarrow \text{SCRA-BGLS.Kg}(1^k)\): \(G_1\) and \(G_2\) are two (multiplicative) cyclic groups of prime order \(p, g_1\) and \(g_2\) are generators of \(G_1\) and \(G_2\), respectively. \(G_T\) is an additional group such that \(|G_1| = |G_2| = |G_T|\). \(\gamma\) is a bilinear pairing \(\hat{\gamma} : G_1 \times G_2 \to G_T\) such that \(i) \text{Bilinear: for all } u \in G_1, v \in G_2, \hat{\gamma}(u^a, v^b) = \hat{\gamma}(u, v)^{ab}; ii) \text{Non-degenerate: } \hat{\gamma}(g_1, g_2) \neq 1\) (please refer to [7] for details). Finally, \(H' : \{0,1\}^* \to G_1\) is a Full Domain Hash [19] modeled as RO [4].
1. Set \(sk^i \leftarrow x \text{ and } PK^i \leftarrow g_2^x \in G_2\) [7], where \(x \overset{\$}{\leftarrow} \mathbb{Z}_p.\)
2. Compute \(\gamma_{i,j} \leftarrow H'(m_{i,j})G_1, i = 1, \ldots, L, j = 0, \ldots, 2^b - 1,\) set \((\Gamma, sk, PK)\) as in Algorithm 1.

1. \(\sigma \leftarrow \text{SCRA-BGLS.Sig}(m, sk)\): Execute Algorithm 1 SCRA.Sig Step 1-2, and obtain \(\gamma_i\) of \(m_i^i\) from \(\Gamma, i = 1, \ldots, L.\) Compute \(s \leftarrow \prod_{i=1}^L (\gamma_i) \in G_1.\) Set \(\sigma\) as in Algorithm 1 SCRA.Sig Step 3.

1. \(\{0,1\} \leftarrow \text{SCRA-BGLS.Ver}(m, \sigma, PK)\): Execute Algorithm 1 SCRA.Ver Step 1-2. In Step 3 (i.e., aggregate signature verification), if \(\hat{\gamma}(s, g_2) = \prod_{i=1}^L \hat{\gamma}(H'(m_i^i), g_2^x) \in G_2,\) return 1, else return 0.

**Note:** We implement SCRA-BGLS on an elliptic curve \(E,\) in which modular exponentiation and multiplication correspond point scalar multiplication and point addition on \(E\) [17], respectively.

**SCRA-NTRU** is based on NTRU aggregate signature [15]. Note that SCRA-NTRU achieves the highest signing efficiency among all instantiations (it is even more efficient than SCRA-C-RSA at the signer side). It also has a low end-to-end delay, which is comparable to SCRA-C-RSA but slightly less efficient, since NTRU aggregate signature verification algorithm in [15] is less efficient than that of SCRA-NTRU and a low end-to-end delay but with a larger signature size. SCRA-NTRUPASS is based on the PASS [18] signature scheme. It provides similar performance to SCRA-NTRU in terms of both latency and storage but the lattice based scheme is more practical to use. This means that
Algorithm 4 SCRA instantiation with lattice-based sequential aggregate signatures [15]: SCRA-NTRU

\[
\begin{align*}
(sk, PK) & \leftarrow \text{SCRA-NTRU.Kg}(1^n) \\
\text{Set } f_A : B_n \rightarrow R_n \text{ as a family of preimage-sampleable trapdoor function NITRU.Sign [27], where } P \leftarrow A = g/f \in R_q^X, \\
\text{sk}' & \leftarrow T = \left[ \begin{array}{c} f \\ g \\ F \\ O \end{array} \right]. T \text{ is the trapdoor and } B_B \text{ is the domain of } f_A, \omega_i \text{ represents the list of i partial aggregate signatures.}
\end{align*}
\]

1. \( \sigma \leftarrow \text{SCRA-NTRU.Sig}(m, sk) \): Execute Algorithm 1 SCRA.Sig Step 1-2, and obtain \( \gamma^i \) of \( m^i \) from \( \Gamma \), \( i = 1, \ldots, L \). Compute \( s \leftarrow \text{AggSign}(T, \gamma_i, \omega_{i-1}) \) for \( i = 1, \ldots, L \). Set \( \sigma \) as in Algorithm 1 SCRA.Sig Step 3.

2. \( \{0,1\} \leftarrow \text{SCRA-NTRU.Ver}(m, \omega, PK) \): Execute Algorithm 1 SCRA.Ver Step 1-2. In Step 3 (i.e., aggregate signature verification), if \( \text{Agg.Verify}(A, m, s, \{\omega^i\}) \), return 1, else return 0.

Algorithm 5 SCRA instantiation with lattice-based sequential aggregate signatures [15]: SCRA-NTRUPASS

\[
\begin{align*}
(sk, PK) & \leftarrow \text{SCRA-NTRUPASS.Kg}(1^n) \\
\text{Set } f_A : B_n \rightarrow R_n \text{ as a family of preimage-sampleable trapdoor function PASSSign [18], where } sk' \text{ is a polynomial } L^\infty \text{ norm equal to } 1 \text{ (coefficients are chosen independently from the set } [-1, 0, 1]), P \leftarrow A = F_{T, \gamma} \cdot sk', T \text{ is the trapdoor and } B_B \text{ is the domain of } f_A, \omega_i \text{ represents the list of i partial aggregate signatures.}
\end{align*}
\]

1. \( \sigma \leftarrow \text{SCRA-NTRUPASS.Sig}(m, sk) \): Execute Algorithm 1 SCRA.Sig Step 1-2, and obtain \( \gamma^i \) of \( m^i \) from \( \Gamma \), \( i = 1, \ldots, L \). Compute \( s \leftarrow \text{AggSign}(T, \gamma_i, \omega_{i-1}) \) for \( i = 1, \ldots, L \). Set \( \omega \) as in Algorithm 1 SCRA.Sig Step 3.

2. \( \{0,1\} \leftarrow \text{SCRA-NTRUPASS.Ver}(m, \sigma, PK) \): Execute Algorithm 1 SCRA.Ver Step 1-2. In Step 3 (i.e., aggregate signature verification), if \( \text{Agg.Verify}(A, m, s, \{\omega^i\}) \), return 1, else return 0.

SCRA-NTRUPASS is secure for usage even with smaller parameters as it is based on the partial Fourier recovery problem rather than the approximate CVP problem for SCRA-NTRU.

IV. Security Analysis

We now present our security analysis for SCRA schemes.

Theorem 1 SCRA is \((t, q_s, \epsilon)-EU-CMA secure, if the underlying ASig is \((t', q_s, \epsilon)-IA-EU-CMA secure, where \( t' = O(t) + (\text{ASGN } + \text{RO}();) = q_s \). RO() and ASGN denote the cost of random oracle invocation and aggregate signature generation, respectively.

Proof: Suppose that A breaks \((t, q_s, \epsilon)-EU-CMA secure SCRA. We construct a simulator \( \mathcal{F} \), which breaks \((t', q_s, \epsilon)-IA-EU-CMA secure ASig by using A as a subroutine with the experiment below:

Setup: \( \mathcal{F} \) is provided with two oracles as below:

1. A random oracle \( h \leftarrow \text{RO}(m) \), which returns \( h \leftarrow \{0,1\}^d \) if \( m \in \{0,1\}^* \) has not been queried before, else it returns the same answer \( h \) for the given \( m \). That is, the cryptographic hash function \( H \) in SCRA is modeled as a random oracle.

2. A signature oracle \( s \leftarrow \text{Osk}(m) \) as in \( IA-EU-CMA \) experiment (i.e., Definition 5). That is, given a query \( m = (m_1, \ldots, m_L) \), \( \mathcal{O} \) returns an aggregate signature \( s \leftarrow \text{ASig} \cdot \text{Agg}(\gamma_1, \ldots, \gamma_L) \), as described in [3], that is secure in the random oracle model [4]. Let \( \text{Agg.Sign} \) and \( \text{Agg.Verify} \) be functions as described in [15].

For each query \( j \), \( \mathcal{F} \) performs the following operations:

1. \( \mathcal{F} \) queries RO() on \( (m_j || r_j) \), and receives an answer as \( (\tilde{m}_j, \ldots, \tilde{m}_{j,L}) \leftarrow \text{RO}(m_j || r_j) \), where \( r_j \leftarrow \{0,1\}^k \) such that \( \{\tilde{m}_j, \ldots, \tilde{m}_{j,L}\} = b_{j} \cdot \mathcal{A} \) (as in Algorithm 1 SCRA.Kg Steps 2-3 and SCRA.Sig Step 1).

2. \( \mathcal{F} \) queries \( s_j \leftarrow \text{Osk}(\tilde{M}_j) \), where \( \mathcal{M}_j = \{l \mid m_{j,l} \in \{0,1\}^* \text{ for } l = 1, \ldots, L \} \). \( \mathcal{F} \) sends \( s_j \leftarrow (s_j, r_j) \) to A (as in Algorithm 1 SCRA.Kg Steps 2-3).

For each query \( j \), \( \mathcal{F} \) outputs a forgery \((m^*, \sigma^*) = (s^*, r^*) \) and wins the EU-CMA experiment, if (i) \( \mathcal{F} \text{ SCRA.Ver}(m^*, \sigma^*, PK) = 1 \) and (ii) \( m^* \notin \{m_1, \ldots, m_q\} \). If A loses in the EU-CMA experiment, then B also loses in the IA-EU-CMA experiment and aborts. Otherwise, \( \mathcal{F} \) returns a forged signature as \( (s^*, \sigma^*) \), the \( \mathcal{M}_j = \{l \mid m_{j,l} \in \{0,1\}^* \text{ for } l = 1, \ldots, L \} \) such that \( (\tilde{m}_j^*, \ldots, \tilde{m}_{j,L}^*) \leftarrow \text{RO}(m^* || r^*) \) (as in Query phase Step 1). \( \mathcal{F} \) checks the forgery conditions for
IA-EU-CMA experiments as in Definition 5 as follows:

1) Validity: Given that SCRA.Ver$(m^*,\sigma^*,PK)$ = 1 holds, ASig.Ver$(\tilde{M}^*,s^*,PK') = 1$ also holds. Therefore, ASig forgery is valid.

2) Non-triviality: $F$ checks if $\tilde{M}^* \not\subseteq \{\tilde{M}_1,\ldots,\tilde{M}_q\}$ holds.

This implies the conditions below:

a) At least one data item $(j||\tilde{m}^*_j||P) \in \tilde{M}^*$ has never been queried to $O$ (i.e., the forgery condition 3.i in Definition 5). Hence, the IA-EU-CMA-secure ASig is broken.

b) The signature extraction occurs by Definition 4 condition 2-3 as $\exists I' \subseteq \{1,\ldots,\eta\} : \tilde{M}^* \subseteq \{k \in I' : \tilde{M}_k\}$ (i.e., the forgery condition 3.ii in Definition 5). This implies that $\tilde{M}^*$ as a batch query has never been queried to $O$. At the same time, each data item $(j||\tilde{m}^*_j||P) \in \tilde{M}^*$ has been queried as a part of a batch query $k \in I' \tilde{M}_k$, and $s^*$ is the aggregation of their corresponding individual signatures (i.e., individual signatures have been extracted and combined as in Definition 4). Finally, the signature extraction is non-trivial since $\tilde{M}^*$ is comprised of $L$ data items and therefore it cannot be a trivial combination of previously asked batch queries. Hence, the IA-EU-CMA-secure ASig is broken.

If the above conditions hold, $F$ wins in the IA-EU-CMA experiment against ASig. Otherwise, $F$ aborts.

The probability that $F$ wins in the IA-EU-CMA experiment is identical to that of $A$ winning in the $EU$-CMA experiment. Note that $H$ is modeled as a random oracle, and therefore the probability that $H$ is not target collision-resistant or subset-resilient [34] is a negligible probability in terms of $\kappa$ (i.e., $1/2^{\kappa}$). For each query of $A$, $F$ performs a query to $RO(.)$ and another query to $O$. Hence, the execution time of $F$ is that of $A$ plus $(ASN + RO(.) \cdot q_s)$. \hfill \Box

We now prove that the SCRA-C-RSA and SCRA-BGLS schemes are secure in Theorem 2 and Theorem 3, respectively. Remark that, for the sake of brevity, we refer to the generic proof in Theorem 1 for common steps, and only emphasize the scheme-specific steps in these theorems.

**Theorem 2** SCRA-C-RSA is $(t,q_s,\epsilon)$-EU-CMA secure, if the underlying C-RSA is $(t',q_s,\epsilon)$-IA-EU-CMA secure, where $t' = O(t) + [RO(.) + L \cdot (\text{Exp} + \text{Mul} + H^2)] \cdot q_s$, RO$(.)$, $\overline{H}$, Exp and Mul denote the cost of random oracle invocation, hash function $H'$, modular exponentiation, and multiplication under modulo $n$, respectively.

**Proof:** Suppose that $A$ breaks $(t,q_s,\epsilon)$-EU-CMA secure SCRA. We construct a simulator $F$, which breaks $(t',q_s,\epsilon)$-IA-EU-CMA secure C-RSA by using $A$ as a subroutine as follows:

**Setup:** $F$ is given RO$(.)$ and $O_{sk}$ as in Theorem 1 Setup Phase. By Algorithm 2, $(sk' = (n,d),PK' = (n,e))$ and hash function used by $O$ is $H'$ : $\{0,1\}^* \rightarrow Z_n$ that behaves as a RO. $F$ gives PK $\leftarrow (PK',P)$ to $A$, where $P \subseteq \{0,1\}^d$

as in Algorithm 2.

**Queries:** $A$ queries $F$ on $m_j \in \{0,1\}^*$ for $j = 1,\ldots,q_s$. For each query $j$, $F$ queries $O$ on $(1||\tilde{m}_{j,1}||P,\ldots,L||\tilde{m}_{j,L}||P) \leftarrow RO(m_j||r_j)$ as in Theorem 1 Query Phase and gets $s_j \leftarrow \prod_{i=1}^{L} H'(i||\tilde{m}_{j,i}||P) \mod n$ as in Algorithm 2. $F$ returns $(s_j,r_j)$. $\Box$

$F$ outputs a forgery $(m^*,\sigma^* = (s^*,r^*)) and checks the $EU$-CMA experiment winning conditions (i)-(ii) as in Theorem 1 Forgery Phase. If they hold, then $F$ returns a C-RSA forgery for IA-EU-CMA experiment as $(\tilde{M}^*,s^*)$ as in Theorem 1 Forgery Phase and proceeds as follows:

1) Validity: SCRA.Ver$(m^*,\sigma^*,PK) = 1$ implies $(s^*)^e = \prod_{j=1}^{L} H'(i||\tilde{m}^*_j||P) \mod n$ holds. Therefore, C-RSA forgery is valid.

2) Non-triviality: $F$ checks if one these conditions hold:
   i) At least one data item $(j||\tilde{m}^*_j||P) \in \tilde{M}^*$ has never been queried to $O$. This implies that IA-EU-CMA-secure C-RSA is broken, since by the validity condition, there is a signature as $s' = H'(j||\tilde{m}^*_j||P)^d \mod n$, which was not obtained from $O$. ii) The signature extraction occurs as defined in Theorem 1 Non-triviality condition (b). That is, individual signatures $s_j = H'(j||\tilde{m}^*_j||P)^d \mod n$ have never been individually queried to $O$, but all were part of a batch query $k \in I' \tilde{M}_k$. This implies IA-EU-CMA-secure C-RSA is broken by the signature extraction argument as in [30], [42] (see Section II-C). The non-triviality holds as in Theorem 1.

The success probability is as in Theorem 1 and the probability that $H'$ produces a collision is $1/2^{n/2}$. For each query of $A$, $F$ performs a query to $RO(.)$ and $O$, which requires a $H'$ computation, followed by an exponentiation/multiplication under $n$ for each item in $(1||\tilde{m}_{j,1}||P,\ldots,L||\tilde{m}_{j,L}||P)$. Hence, the execution time of $F$ is that of $A$ plus $[RO(.) + L \cdot (\text{Exp} + \text{Mul} + \overline{H}^2)] \cdot q_s$. \hfill \Box

**Theorem 3** SCRA-BGLS is $(t,q_s,\epsilon)$-EU-CMA secure, if the underlying BGLS is $(t',q_s,\epsilon)$-IA-EU-CMA secure, where $t' = O(t) + [RO(.) + L \cdot (\text{Exp} + \text{Mul} + \overline{H}^2)] \cdot q_s, RO(.)$, $\overline{H}$, Exp and Mul denote the cost of random oracle invocation, hash function $H'$, modular exponentiation and multiplication in $G_1$, respectively.

**Proof:** Suppose that $A$ breaks $(t,q_s,\epsilon)$-EU-CMA secure SCRA. We construct a simulator $F$, which breaks $(t',q_s,\epsilon)$-IA-EU-CMA secure BGLS by using $A$ as a subroutine with the experiment below:

**Setup:** $F$ is given RO$(.)$ and $O_{sk}$ as in Theorem 1 Setup Phase. By Algorithm 3, $(sk' = (x,PK' = g_x^2 \in G_2)$ and $H' : \{0,1\}^* \rightarrow G_1$ is a RO. $F$ gives PK $\leftarrow (PK',P)$ to $A$ as in Algorithm 3.

**Queries:** $A$ queries $F$ on $m_j \in \{0,1\}^*$ for $j = 1,\ldots,q_s$. For each query $j$, $F$ queries $O$ on $(1||\tilde{m}_{j,1}||P,\ldots,L||\tilde{m}_{j,L}||P) \leftarrow RO(m_j||r_j)$ as in Theorem
1 Query Phase and gets \( s_j \leftarrow \prod_{i=1}^{L} h'(\hat{m}_{i,j})^x \in G_1 \) as in Algorithm 3. \( F \) returns \( \sigma_j = (s_j, r_j) \).

**Forgery**: \( A \) outputs a forgery \((m^*, \sigma^* = (s^*, r^*))\) and checks E\(U\)-CMA experiment winning conditions (i)-(ii) as in Theorem 1 **Forgery Phase**. If they hold then \( F \) returns a BGLS forgery for IA-EU-CMA experiment as \((M^*, s^*)\) as in Theorem 1 **Forgery Phase** and proceed as follows:

1. **Validity**: SCRA.Ver \((m^*, \sigma^*, PK) = 1\) implies that that \( e(s^*, g_2) = \prod_{i=1}^{L} e(h'(i)[\hat{m}_{i}'], P), g_2^j \in G_2 \). Therefore, BGLS forgery is valid.

2. **Non-triviality**: \( F \) checks if one of these conditions hold. (i) At least one data item \((j)[\hat{m}_j'] \in M^*\) has never been queried to \( O \). That is, IA-EU-CMA-secure BGLS is broken, since by validity condition, there is a signature as \( e(s^*, g_2) = e(h'(j)[\hat{m}_j'], P), g_2^j \in G_2 \), which was not obtained from \( O \). (ii) The signature extraction occurs as defined in Theorem 1 **Non-triviality** condition (b). That is, individual signatures \( e(s^*, g_2) = e(h'(j)[\hat{m}_j'], P), g_2^j \in G_2 \), \( j = 1, \ldots, L \) have never been individually queried to \( O \), but all were part of a batch query \( k \in \Pi \hat{M}_k \). This implies IA-EU-CMA-secure BGLS is broken by the signature extraction argument as in [10] (see Section II-C). The non-triviality of signature extraction holds as in Theorem 1.

The success probability analysis is as in Theorem 1 and the probability that \( H' \) produces a collision is \( 1/2^{[L^L]} \). For each query of \( A \), \( F \) performs a query to \( RO(\cdot) \) and an another query to \( O \), which requires a \( H' \) computation, followed by an exponentiation and multiplication in \( G_1 \) for each item in \((1)[\hat{m}_{i,j}], P, \ldots, L)[\hat{m}_{i,j}] \). Hence, the execution time of \( F \) is that of \( A \) plus \( t^* = O(t) + [RO(\cdot) + L \cdot (Expn + Muln + H')] \cdot q_s \).

**Remark 1** The formal proof of SCRA-NTRU and SCRA-NTRUPASS follow a similar logic and therefore will not be repeated here. At the same time, we note that despite the existence of an A-EU-CMA analysis, IA-EU-CMA proof and analysis for signature extraction argument are not currently available for NTRU signatures. Hence, a full formal reduction requires this gap to be filled first, which is out of the scope of this paper.

### V. Performance Analysis and Comparison

In this section, we present the performance results of our experiments. We first compare the results of the SCRA with the state-of-the-art algorithms on a modern powerful CPU. We then provide results for the GPU implementations of SCRA-C-RSA and SCRA-NTRU as compared to their CPU counterparts. For the GPU, we used an Nvidia Tesla K40c card, which is comprised of 2880 computing cores with 12GB of GDDR5 device memory and 288GB/sec memory bandwidth. Our base system is equipped with an Intel Core i7-6700K 4.0GHz Quad-Core Processor and 16GB DDR4 2400 MT/s. This infrastructure represents a datacenter setting. We also implemented SCRA on a System-on-Chip (SoC). We used an Nvidia Tegra K1 SoC, which has a 4-Plus-1 quad-core ARM Cortex A15 CPU with clock rate of 2.3 Ghz and an embedded GPU with 192 computing cores. Such SoCs represent smaller scale systems that are widely used in IoT deployments. We open sourced the source code for the research and academic community to use and evaluate1.

We summarize the results in Table 1. Table I also provides the implementation details, parameters and key/table sizes. Table I shows the clear superiority of SCRA in terms of signature generation efficiency and end-to-end cryptographic delay (i.e., the sum of signature generation and verification times) using a powerful CPU. That is, the signature generation of SCRA instantiations are 24, 18 and 516 times faster than their non-SCRA counterparts for RSA, BGLS, and NTRU, respectively. This indicates that SCRA is an ideal choice for a very high-throughput signature generation, especially for resource-limited devices in IoT deployments. Similarly, SCRA-C-RSA and SCRA-NTRU offer 18 and 7 times lower end-to-end crypto delay compared to RSA and NTRU, respectively, making them ideal choices for time-critical authentication.

In addition to their computational efficiency, the SCRA schemes are also compact, since the signature and public key sizes remain the same with their base signature scheme (the transmission of \(|\sigma| = \kappa\) is negligible). By comparing to each other, SCRA-C-RSA achieves the lowest end-to-end delay with a moderate signature size (e.g., 256 bytes), while SCRA-BGLS offers the smallest signature (20 bytes) but the highest end-to-end delay. SCRA-NTRU has the lowest signing delay (0.0018 msec), low end-to-end delay but with large signatures (e.g., 1587 bytes). Note that all SCRA schemes require storing a pre-computed table \( \Gamma \), which introduces a constant-size extra storage overhead at the signer side e.g., 160 KB, 2 MB and 12.33 MB for SCRA-BGLS, SCRA-C-RSA and SCRA-NTRU respectively. This signer storage is plausible even for some embedded devices (e.g., Raspberry PI 2 [1]), and negligible for vehicular networks. Moreover, recall that, unlike offline-online signatures, the signer overhead of SCRA is constant and it does not require to regenerate tokens.

The offline stages of the algorithms take fairly minimal times of 2.45, 8.65 and 12.83 seconds for SCRA-BGLS, SCRA-C-RSA and SCRA-NTRU respectively. The offline stage will only be required to execute once during the system deployment.

**Space versus execution time**: We have a trade-off between the space taken to store the signatures and the execution time of the signing and the verification stages. As described in 1, an \( d \)-bit hash output can be interpreted as integers \( (j_1, \ldots, j_k) \), where each \( j_i \) is a \( b \)-bit integer such that \( b \cdot L = d \). The total number of signatures that need to be calculated and stored in the offline stage of an algorithm is thus \( L \cdot 2^b \). The total storage cost is thus \( L \cdot 2^b \cdot S \) where \( S \) is the size of one signature. This also implies that the number of aggregations to be performed during the online phase increases linearly with \( L \). Table II provides for the SHA-256 hashing scheme various values of \((L, b)\) parameters and corresponding size of the signature table.

---

1. [https://github.com/Ipapapa/HWAccelerated-Crypto]
Table II: The storage space of signature table and the number of aggregations required for various $L$ and $b$ values.

<table>
<thead>
<tr>
<th>$L$ (# bit chunks)</th>
<th>$b$ (# of bits # in a chunk)</th>
<th>SCRA-RSA (KB)</th>
<th>SCRA-BGLS (MB)</th>
<th>SCRA-NTRU (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>16</td>
<td>20480</td>
<td>256</td>
<td>1578</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
<td>160</td>
<td>2</td>
<td>12.32</td>
</tr>
<tr>
<td>64</td>
<td>4</td>
<td>20</td>
<td>0.25</td>
<td>1.54</td>
</tr>
<tr>
<td>128</td>
<td>2</td>
<td>10</td>
<td>0.125</td>
<td>0.77</td>
</tr>
<tr>
<td>256</td>
<td>1</td>
<td>10</td>
<td>0.125</td>
<td>0.77</td>
</tr>
</tbody>
</table>

One may observe that the smallest storage overhead can be attained with $(L = 256, b = 1)$, wherein we store a signature for every bit in the hash output domain. However, this requires $L = 256$ signature aggregations in the online signature generation phase (e.g., 256 modular multiplications for SCRA-C-RSA), which may not be computationally efficient. Another end of the trade-off is $(L = 16, b = 16)$, wherein only $L = 16$ online aggregations are required during the online phase. However, this requires substantially larger table sizes, which may be suitable for some real-life applications. We observed that $(L = 32, b = 8)$ offers a highly favorable overall performance/storage performance, as shown in Table I and Table II. The size of the pre-computed tables for SCRA-BGLS, SCRA-C-RSA and SCRA-NTRU is 160 KB, 2 MB and 12.33 MB, respectively, for signature-sizes of 20, 256 and 1578 bytes, respectively. This will require 32 signatures to be aggregated in the online-signing phase of each SCRA scheme.

VI. HARDWARE ACCELERATION OF SCRA

To accelerate SCRA, we leveraged the parallel processing and optimization capabilities of GPUs both on server and embedded in the SoCs. We have introduced several optimizations to parallelize the individual steps of SCRA algorithms. We used optimizations specific to the architecture of the GPU to harness the vast amount of available lightweight cores [11].

A. Accelerating SCRA-C-RSA with GPUs

- **SCRA RSA - Server:** In the offline signature stage, for 8192 messages, we achieve x1.3 times more throughput with our GPU optimizations compared to the CPU only implementations. In the online signature stage, we achieve significantly high throughput gains, which can reach up to x5.3 times. In the verify stage, the gain is around x4.2 times. These results are reported in Figures 2 and 3. In terms of execution time, the GPU can process a message in 0.367, 0.022, 0.031 milliseconds for the offline, online and verify stages of the algorithm, respectively. This is approximately x1.31, x5, x4 times faster than the corresponding CPU execution times. The GPU gives a worse performance than the CPU if a very small number of messages are processed. This is mainly due to the low clock speeds of the GPU cores as compared to the CPU and also due to the time to copy the data from the CPU memory to the GPU memory and vice-versa. Our experiments show that the online signature
and signature verification stages are executed faster in the GPU than in the CPU for message batches greater than 128 and 256, respectively.

- **SCRA RSA - SoC:** In the offline signature stage, for 8192 messages, we achieve x3.2 times higher throughput with our GPU optimizations compared to CPU only implementations. In the online signature stage, we achieve high throughput gains up to x5.2 times. In the verify stage, the gain is around x4.8 times. These results are reported in Figures 4 and 5.

Below we describe the techniques we adopted to achieve some of the performance speedups shown by above experiments.

**Chinese Remainder Theorem (CRT):** We leverage CRT [28] to accelerate **SCRA** on GPUs. We split a \( k/2 \) bit signature \( \sigma \) into two \( k/2 \) bit signatures \( \sigma_1 \) and \( \sigma_2 \). \( \sigma_1 = M^d \mod p \mod \sigma \), where \( M \) is the message and \( (p, q) \) are the primes used. Then, we use the mixed radix conversion algorithm [22] to combine the two parts and recover the signature \( \sigma \) as \( \sigma = \sigma_1 + [r(\sigma_2 \cdot (q^{-1} \mod p))q] \). These two parts are processed on separate threads in the GPU, which is significantly faster than the \( k \)-bit modular exponentiation.

**Montgomery multiplication:** The modular multiplication is inefficient in the GPUs since it requires a trial division to determine the result and is not parallelizable. The Montgomery multiplication is suitable for implementation in a GPU, since it does not require a trial division and can be implemented in parallel on separate words of the message. That is, given \( a \cdot b \mod n \), we first find two integers \( r^{-1} \) and \( n' \) using the Extended Euclidean Algorithm such that \( rr^{-1} - nn' = 1 \). We then transform \( a = ar \mod n \) and \( b = br \mod n \). Later, we compute \( a \cdot b \mod n \) by using Montgomery reduction [28].

**Batch processing:** The crypto operations for multiple messages are performed concurrently in the GPU. This requires that a batch of messages be passed to the GPU, instead of a single message.

**Breakup of components into words:** To optimize the throughput on the GPU, each message component is divided into words of size 32/64 bits, depending on the GPU capabilities. Each operation being run on a single thread is run over words rather than over entire message components. We use standard multi-precision algorithms [12] to represent and perform operations between large integers.

**GPU warp size utilization:** Warps are set of threads (generally 32) that are considered as one single execution unit inside a CUDA block. To gain maximum throughput from the GPU, it is necessary to attain the maximum number of active warps per streaming multiprocessor which is 64 in our case. We achieve this by adjusting the number of threads per block to the optimal value.

**Memory latency vs GPU occupancy:** The size of the shared memory can limit the number of active warps on the GPU at a particular point in time by reducing the occupancy of the Streaming Multiprocessors (SM). The other limiting factor in the performance output is the number of reads and writes on the global memory on the device. We identified a balance between the SM occupancy and the global memory read/write latency by testing various permutations of memory allocations among the shared and global memory.

**Constant Length Non-zero Window Technique:** We scan the bits of the exponent from the least significant bit to the most significant bit. At each step, we compute a zero window or a non-zero window [23]. With the binary square-and-multiply method, we can process these windows and reduce the number of modular multiplications, making the exponentiation algorithm faster.

### B. Accelerating SCRA-NTRU with GPUs

- **SCRA NTRU - Server:** In the online signature stage, for 4096 messages, we achieve x0.79 times more throughput with our GPU optimizations compared to CPU only implementations. In the verify signature stage and offline stage, we achieve
high throughput gains up to x6.5 and x30 times respectively. These results are reported in Figures 6 and 7, respectively. The cryptographic operations for multiple messages are performed concurrently on the GPU. This requires that a batch of messages be passed to the GPU, instead of a single message for the signing and verification stage. We do not employ the GPU for the online stage of SCRA-NTRU because the signature aggregation technique is computationally expensive and deploying it on a GPU core provides little performance benefit. Due to these reasons, the CPU performs better than the GPU during the online stage of the protocol.

- SCRA NTRU - SoC: In the online signature stage, for 1024 messages, we achieve x0.81 times more throughput with our GPU optimizations compared to CPU only implementations. In the verify sign stage and offline stage, we achieve high throughput gains up to x6.65 and x17.7 times respectively. These results are reported in Figures 8 and 9 respectively.

We summarize below the optimizations that have resulted in the performance gains shown by the previous experiments.

**Batch Processing**: Message components are processed in batches as in Section VI-A. As mentioned before, we do not use the GPU for the online stage of SCRA-NTRU.

**Convolution Operations**: The convolution operations in the NTRU in the signing and verification phase are accelerated by employing GPUs. The convolution operation between two $n$ bit polynomials is divided into $n$ cores for each operation where each core is responsible for calculating one bit of the resulting polynomial.

**Fourier Transformations**: Implementing the Fourier transformation on GPUs further accelerates the signing, verification and offline stages. Due to the use of faster convolution and Fourier transformation operations on GPUs, the verify stage of the protocol on GPUs is significantly faster than on CPUs.

### VII. Conclusion

In this paper, we developed a new series of delay-aware digital signatures for time-critical applications, which we refer to as **Structure-Free Compact Authentication (SCRA)**. SCRA can transform any secure aggregate signature into a signer efficient signature via a novel constant-size pre-computation strategy. We proposed several instantiations of SCRA schemes based on Condensed-RSA, BGLS, and NTRU signatures, each offering a unique computation time, key and signature size trade-offs. Our implementations and performance comparison with the existing alternatives show that the SCRA schemes achieve significantly faster signature generation and lower end-to-end delay. We also formally proved that SCRA schemes are secure (in ROM). Finally, we pushed the performance of SCRA schemes to their edge by fully implementing them on server-grade GPUs and SoCs, which indicated significant performance gains. All these properties make the SCRA schemes a suitable alternative for delay-aware authentication for time-critical applications.

### References


Atila Altay Yavuz is an Assistant Professor in the School of Electrical Engineering and Computer Science, Oregon State University (August 2014). He was a member of the security and privacy research group at the Robert Bosch Research and Technology Center North America (2011-2014). He received his PhD degree in Computer Science from North Carolina State University in August 2011. He received his MS degree in Computer Science from Bogazici University (2006) in Istanbul, Turkey. He is broadly interested in design, analysis and application of cryptographic tools and protocols to enhance the security of computer networks and systems. Atila Altay Yavuz is a recipient of NRC-CREFA (2017). His research on privacy enhancing technologies (searchable encryption) and intra-vehicular network security are in the process of technology transfer with potential world-wide deployments. He has authored more than 40 research articles in top conferences and journals along with several patents. He is a member of IEEE and ACM.

Aman Mudgerikar is currently pursuing his Ph.D. degree with major in Information Security at the Computer Science Department at Purdue University, West Lafayette, IN, USA. He received his bachelor degree in information and communication technology from DA-IICT, India and his master degree in information security from CERIAS, Purdue University. His current research interests include cryptography, intrusion detection systems and network security.

Ankush Singla is a PhD student in the Computer Science department at Purdue University specializing in Information Security and Assurance. He has worked on projects ranging from “Hardware Accelerated Authentication” and “Centralized Lightening management for Energy Saving”. His current research interests include Blockchain usage for IoT authentication and Certificateless cryptography.

Ioannis Papapanagiotou holds a dual major Ph.D. degree in Computer Engineering and Operations Research from NC State University. He is currently an architect at Netflix, a research assistant professor at the University of New Mexico, and a graduate faculty at Purdue University. Ioannis has served in the faculty ranks of Purdue University (tenure-track) and NC State University. Between 2010-2013, he was working for IBM’s CTO office. He has been awarded the NetApp faculty fellowship and established an Nvidia CUDA Research Center at Purdue University. Ioannis has also received the IBM Ph.D. Fellowship, Academy of TV the PhD Fellowship for his Ph.D. research, and best paper awards in several IEEE conferences for his academic contributions. Ioannis has authored approximately 40 research articles and 10 patent disclosures. Ioannis has also served as a TPC chair in a number of IEEE conferences. Ioannis is a senior member of IEEE and ACM.

Elisa Bertino is professor of computer science at Purdue University, and serves as Director of the CyberSpace Security Lab (CyberSSLab). Prior to joining Purdue in 2004, she was a professor and department head at the Department of Computer Science and Communication of the University of Milan. She has been a visiting researcher at the IBM Research Laboratory (now Almaden) in San Jose, at the Microelectronics and Computer Technology Corporation, at Rutgers University, at Telcordia Technologies. Her recent research focuses on database security, digital identity management, policy systems, and security for web services. She is a Fellow of ACM, of IEEE, and AAAS. She received the IEEE Computer Society 2002 Technical Achievement Award, the IEEE Computer Society 2005 Kanai Award, and the ACM SIGSAC Outstanding Contributions Award. She is currently serving as EiC of Transactions on Dependable and Secure Computing.