AN ABSTRACT OF THE THESIS OF

Muslum Ozgur Ozmen for the degree of Master of Science in Computer Science presented on May 15, 2018.

Title: Efficient Public Key Cryptography Frameworks for Emerging IoT Applications

Abstract approved: ____

Attila A. Yavuz

Internet of Things (IoT) is an integral part of application domains such as smart-home, digital healthcare, smart grid systems and vehicular networks. Various standard public key cryptography techniques (e.g., key exchange, public key encryption, digital signature) are available to provide fundamental security services for IoTs. However, despite their pervasiveness and well-proven security, they also have been shown to be highly costly for embedded devices in terms of energy and time consumption. These standard techniques introduce high delays that may hinder the safe operation of the IoT applications (e.g., smart grids, vehicular networks). Hence, it is a critical task to improve the efficiency of standard cryptographic services, while preserving their desirable security properties simultaneously.

To address the efficiency of the public key cryptography in IoT setting, we propose (i) a series of algorithmic improvements over key exchange and public key encryption schemes and (ii) an attack and an efficient fix to a real-time digital signature scheme that benefits aggregate signatures.

In this thesis, we first exploit synergies among various cryptographic primitives (key exchange and public key encryption schemes) with algorithmic optimizations to substantially reduce the energy consumption of standard cryptographic techniques on embedded devices. Our contributions are: (i) We harness special pre-computation techniques, which have not been considered for some important cryptographic standards to boost the performance of key exchange, integrated encryption, and hybrid constructions. (ii) We provide self-certification for these techniques to push their performance to the edge. (iii) We implemented our techniques and their counterparts on 8-bit AVR ATmega 2560 and evaluated their performance. We used microECC library and made the implementations on NIST-recommended secp192 curve, due to its standardization. Our experiments confirmed significant improvements on the battery life (up to 7$\times$) while preserving the desirable properties of standard techniques. Moreover, to the best of our knowledge, we provide the first open-source framework including such set of optimizations on low-end devices.

Delay-aware signatures also play an important role to provide authentication for critical IoT applications. A recent attempt to derive signer-efficient digital signatures from aggregate signatures was made in a signature scheme referred to as Structure-free Compact Rapid Authentication (SCRA) (IEEE TIFS 2017). In this thesis, we show that SCRA generic design leaks information about the private keys. For pqNTRUsign instantiation, this leakage can be exploited to recover all private key components with an overwhelming probability by observing only 8192 signatures. We then propose a new signature scheme that we call as *Fast Authentication with Aggregate Signatures* (FAAS), which can transform any single-signer (k-element extraction secure) aggregate signature scheme into a signer-efficient signature scheme. We develop two efficient instantiations of FAAS, namely, FAAS-C-RSA and FAAS-NTRU, both of which achieve a low end-to-end cryptographic delay while FAAS-NTRU also offers a post-quantum promise. Our experiments confirmed that FAAS instantiations offer very fast signature generation with up to 100× speed improvements over their base schemes. Moreover, FAAS signature generation avoids operations such as exponentiation and Gaussian sampling, and therefore offers an improved side-channel resiliency against attacks targeting these operations. All these desirable properties come with the cost of a larger private key and a slight increase in the signature size.

Efficient Public Key Cryptography Frameworks for Emerging IoT Applications

by

Muslum Ozgur Ozmen

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented May 15, 2018
Commencement June 2018

Master of Science thesis of Muslum Ozgur Ozmen presented on May 15, 2018.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

Muslum Ozgur Ozmen, Author

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to my advisor, Professor Attila A. Yavuz. His continuous support, encouragement and excellent guidance have been essential for me, as a beginner to cryptography field. He was there for me for all the problems I faced during this project.

I would also like to thank my committee members, Professor Mike Rosulek, Professor Yeongjin Jang, Professor Rakesh Bobba and Professor Yelda Turkan for their time and support.

I am deeply grateful to my friends, Rouzbeh Behnia and Thang Hoang, for their valuable discussions, insightful comments, help and time.

I would particularly thank my parents, Hatice and Mehmet Ozmen, and my sister, Zeynep Deniz Ozmen, for their unlimited love and support. I owe a very important debt to them.

Most importantly, I would like to offer my special thanks to my significant other and best friend, Deniz Berfin Karakoc and would like to dedicate this thesis to her. Any of this would not be possible without her continuous support, encouragement and love.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF APPENDIX TABLES

Chapter 1: Introduction

Internet of Things (IoT) is a heterogeneous system comprised of interrelated smart-objects and sensors. Due to IoTs' pervasiveness and impact on the real-life applications, it is critical to guarantee their security. Especially, fundamental security services such as authentication, integrity, and confidentiality are required for any viable IoT.

Although various standard cryptographic techniques exist ([64, 6, 5]), the vast majority of them may not fully meet the needs of IoTs, especially when such systems involve resource-limited devices. In particular, despite the recent progress on the capabilities of off-the-shelf embedded systems (e.g, AVR ATmega 2560), the energy-constraints of such devices still pose a critical limitation.

Below, we discuss the limitations of some alternatives and specify the research gap to be addressed.

Problem Statement and Research Gap: Symmetric primitives are preferred for resource-limited devices due to their computational efficiency, however, Public Key Cryptography (PKC) is also an essential tool for IoTs: (i) Energy efficient PKC is necessary for the management/distribution of symmetric keys in ubiquitous IoT systems. (ii) Symmetric primitives might not be scalable for large-distributed systems [62], while PKC can achieve scalability for large systems. (iii) Symmetric primitives do not offer public verifiability and non-repudiation, which are highly desirable for some

IoT applications such as payment systems, secure audit logging, and digital forensics (medical devices). On the other hand, to pervasively deploy PKC in resource-limited IoT systems, the efficiency of PKC primitives should be substantially improved and optimized.

Many techniques are proposed to improve the efficiency of PKC [6, 5]. Improved standards include key exchange (HMQV [37]), integrated encryption (ECIES [47]) and hybrid constructions (Signcryption [64]). To further improve these techniques, lightweight signatures [22], self-certified key exchange [31], and efficient Elliptic Curve (EC) variants [11, 21] have been introduced. Despite their merits, there is a research gap that prevents the full utilization of performance benefits of these techniques for IoT systems:

(i) The integrated schemes and self-certified constructions have various common operations to be synergized. Yet, these primitives are considered in isolation. (ii) These common operations have the potential to receive significant benefits from special algorithmic optimizations [18], which have not been explored for integrated and self-certified cryptographic techniques. (iii) A comprehensive energy consumption analysis of such improved cryptographic techniques on modern embedded devices are currently missing in the literature. (iv) An open-source framework of energy efficient schemes, specifically for IoT applications for public use is necessary.

Aggregate digital signatures allow multiple signatures to be aggregated into a single signature [16]. Due to their communication efficiency, they have been considered for a wide range of applications, including, but not limited to, secure routing [65], authentication in outsourced databases [42] and secure logging [38].

Aggregate signatures have also been used to achieve efficient signature generation as shown in [60] with Rapid Authentication (RA) scheme. RA exploits the fact that the signature aggregation operation is significantly faster than that of the signature generation for some aggregate signature schemes. This permits a signer to pre-compute a set of individual signatures in the key generation algorithm (offline), and then aggregate a subset of them to efficiently compute signatures in the signature generation algorithm (online). This strategy improves the signature generation efficiency, and therefore is useful for time-critical systems (e.g. smart-grids [2]) that require a low cryptographic delay to operate safely. However, RA requires a one-time signature to be stored for each message to be signed, and also needs messages to be in a pre-defined (fixed length) format.

It is highly desirable to develop fast digital signature schemes that can avoid the storage/re-generation of one-time signatures and the need of a pre-defined message format. One recent attempt to address these issues was proposed in *Structure-free Compact and Rapid Authentication (SCRA)* [61]. In this thesis, we show that, it is a challenging and yet feasible task to create such fast signatures by first mounting an attack to SCRA [61], and then constructing new fast digital signature schemes that can address the aforementioned limitations.

Chapter 2: Low-Cost Standard Public Key Cryptography Services for Wireless IoT Systems

2.1 Our Contributions

Towards filling the aforementioned research gaps, we propose a series of cryptographic optimizations that exploit synergies and algorithmic techniques to enable high efficiency and minimum energy consumption for wireless IoT systems.

- *Improving Battery life with Low Storage Overhead*: One of the costly operations in standard PKC suites is EC scalar multiplication (*Emul*). We observe that it can be significantly accelerated with Boyko-Peinado-Venkatesan (BPV) technique [18], whose potential is not investigated for major cryptographic suites (ECHMQV [37], ECIES [47] and Signcryption [64]). We provide, to the best of our knowledge, the first realization of BPV for these suites on embedded devices. We also present further optimizations that we refer to as Designated BPV (DBPV). Our improved suites achieve significantly lower energy consumption with a small constant-storage overhead. Note that the traditional pre-computation techniques incur linear token storage/re-generation costs (a token per-item), which are not feasible for memory limited IoT devices. Moreover, it is shown in [53] that the re-generation of tokens may require more energy and time than just following the standard protocol.

- *Eliminating Certification Overhead*: In aforementioned cryptographic suites, the sender creates an ephemeral ECDH key to be incorporated in encryption and/or signatures. We notice that by transforming this step into a self-certified ECDH operation, for instance via Arazi-Qi (AQ) [5], it is possible to seamlessly eliminate the verification/transmission overhead introduced by certificates.

- *Integration of Optimizations to Standard Suites*: We identify that self-certification synergizes well with BPV, providing further efficiency gain. Our analysis shows significant performance gains for both fixed key exchange and integrated protocols. With these improvements: (i) Our proposed scheme AQ-BPV achieves almost $3\times$ faster key exchange than ECDH with ECDSA certificate, where the transmission cost of the certificate is also eliminated (see Table 2.2). (ii) Our improved schemes with AQ, BPV and DBPV eliminate the overhead of certificates and improve execution time by up to $7\times$ (see Table 2.3) for integrated schemes such as ECIES and Signcryption.

- *Experimental Evaluation and Open-Source Framework*: We implemented our techniques and their counterparts on an 8-bit ATmega 2560 microcontroller which is widely used in IoT applications due to its flexibility and low-power consumption [55, 56]. Our experiments confirmed that our schemes achieve approximately $7\times$ improvement in terms of battery life and computation time (see Section 2.4). Moreover, to the best of our knowledge, there is no open-source library for these cryptographic suites and the improvements we have adopted to low-end embedded devices. Therefore, we are putting an effort for the adoption of our optimizations and these cryptographic suites by making our implementations open-source.

`https://github.com/ozgurozmen/OptimizedPKCSuite`

Limitations: BPV introduces the storage of a 11.25 KB (constant-size) table, and when DBPV is also utilized, this storage overhead increases to 18.75KB. However, we show that such storage is feasible even to 8-bit devices like ATmega 2560 microcontroller, and provides up to 7× time and energy efficiency. Therefore, we believe it is a useful trade-off. The limitation of AQ protocol (which provides self-certification) is that a key generation center (KGC) needs to calculate and distribute the keys to the nodes. While this approach is certainly feasible to be employed in certain IoT applications (e.g., smart airport/city systems), it may not be for some other applications. As self-certification removes all certification overhead, we believe it is useful to adopt AQ protocol when it is feasible.

Note that our optimizations are not tightly coupled. Therefore, for the applications that are not suitable for AQ protocol, BPV and DBPV still provide significant improvements (vice versa). Moreover, these improvements are achieved by preserving the core operations of the base schemes, so they retain their security properties as well as permitting an easy adoption for real-life applications.

2.2 Preliminaries

We first outline notation in Table 2.1, and then describe building blocks used by our proposed schemes as follows:

Arazi-Qi (AQ) Self-Certified Ephemeral Scheme: Arazi-Qi (AQ) [5] proposed a simple yet efficient self-certified ECDH scheme. During the offline phase, all participants in the system are given a self-certified ECDH private/public key pair by

Table 2.1: Notation followed to describe schemes.

$\mathbf{G}$	Generator group point
q	Order of group
d	*Private Key* of CA
$\mathbf{D}$	*Public Key* of CA where $\mathbf{D} = d \times \mathbf{G}$
x_i	Fixed *Private Key* of Node i
$\mathbf{U}_i$	Fixed *Public Key* of Node i
ID_i	Identification of Node i
m	Message
$\times$	Elliptic Curve Scalar Multiplication

Elliptic Curve (EC) points are shown in **bold**.

a CA. At the online phase, any two entities with valid self-certified key pair can establish a symmetric key without requiring the transmission and verification of ECDH certificates. In Algorithm 1, we outline an ephemeral AQ variant proposed by Hang et. al. in [31], which offers higher security guarantees.

As $x_a \times [H(ID_b||\mathbf{U}_b) \times \mathbf{U}_b + D] = x_b \times [H(ID_a||\mathbf{U}_a) \times \mathbf{U}_a + \mathbf{D}] = x_a \cdot x_b \times \mathbf{G}$ is constant for both nodes (which is the fixed key in AQ [5]), they can store this value and use it in future key exchanges. In the online phase, there are only two *Emul* for each node. This also decreases the bandwidth as U_a and U_b are transferred only once.

Boyko-Peinado-Venkatesan (BPV) Technique [18]: This technique reduces the computational cost of a full scalar multiplication to only a few EC additions with the expense of a small-constant size table storage.

2.3 Proposed Techniques

Our target suites are key exchanges (ECHMQV [37], AQ [5]), and integrated protocols (ECIES [47], Signcryption [64]). Our rationale for selecting these cryptographic suites

Algorithm 1 Ephemeral AQ variant by Hang et. al. [31]

```
$(x_a, \mathbf{U}_a, x_b, \mathbf{U}_b) \leftarrow$ AQ-Hang.Offline (offline calculations performed by CA)
1: $b_a \xleftarrow{\$} \mathbf{Z}_q$, $\mathbf{U}_a \leftarrow b_a \times \mathbf{G}$.
2: $x_a \leftarrow [H(ID_a, \mathbf{U}_a) \cdot b_a + d]$ and repeat 1-2 for node B.
3: Node A $\leftarrow (x_a, \mathbf{U}_a)$, Node B $\leftarrow (x_b, \mathbf{U}_b)$.
$K_{ab} \leftarrow$ AQ-Hang.Online (online calculations)
```

Node A		Node B
$p_a \xleftarrow{\$} \mathbf{Z}_q$		$p_b \xleftarrow{\$} \mathbf{Z}_q$
$\mathbf{E}_a \leftarrow p_a \times \mathbf{G}$		$\mathbf{E}_b \leftarrow p_b \times \mathbf{G}$
	$\xrightarrow{\text{Send } (ID_a, \mathbf{U}_a, \mathbf{E}_a)}$	
	$\xleftarrow{\text{Send } (ID_b, \mathbf{U}_b, \mathbf{E}_b)}$	

Node A: $\mathbf{K}_{ab} = x_a \times [H(ID_b||\mathbf{U}_b) \times \mathbf{U}_b + \mathbf{D}] + p_a \times \mathbf{E}_b$.
Node B: $\mathbf{K}_{ab} = x_b \times [H(ID_a||\mathbf{U}_a) \times \mathbf{U}_a + \mathbf{D}] + p_b \times \mathbf{E}_a$.

Algorithm 2 BPV Generator

```
$\Gamma \leftarrow BPV.Offline(n)$, $n$ : Number of pre-computed pairs.
1: $p_i \xleftarrow{\$} \mathbf{Z}_q$, $\mathbf{P}_i \leftarrow p_i \times \mathbf{G}$, and store pairs $\Gamma = \langle (p_i, \mathbf{P}_i) \rangle_{i=1}^{n}$
$(r, \mathbf{R}) \leftarrow BPV.Online(\Gamma)$
1: Generate a random set $S \subset [1, n]$, where $|S| = k$.
2: $r \leftarrow \sum_{i \in S} p_i$ and $\mathbf{R} = r \times \mathbf{G} = \sum_{i \in S} \mathbf{P}_i$.
```

can be summarized as follows: (i) Although ECDH with certificates is very common in practice (SSL/TLS), it is very costly for IoT systems. Therefore, we improve AQ scheme, a lightweight self-certified key exchange protocol, and ECHMQV scheme as it was standardized in IEEE P1363 [1]. (ii) Integrated schemes provide both authentication and encryption with a less cost than considering these two apart. Selected integrated schemes are also standardized (ECIES - IEEE P1363 [1], Signcryption - ISO/IEC 29150:2011) and extensively used in practice.

- *Seamless Integration of Self-Certification:* These cryptographic techniques require a certificate to be transmitted and verified to ensure the authenticity of the public key(s). We notice that these techniques generate an Elgamal encryption key as an (ephemeral) ECDH key. This key is directly used in ECHMQV and ECIES, and also incorporated into joint signature/encryption in Signcryption. We exploit this common step to enable a self-certification by adopting AQ protocol [5]. This strategy permits us to avoid the transmission and verification of certificates but requires all nodes to receive their key set from CA as required by AQ protocol.

- *Constant Size Pre-computation:* Traditional pre-computation techniques store a set of pairs $\langle r_i, r_i \times \mathbf{G} \rangle_{i=1}^{N}$ to avoid online scalar multiplications, which incurs a linear memory overhead. Moreover, once these tokens are depleted, the device must re-generate them, which is highly costly [53]. Hence, these techniques are not suitable for battery-limited IoT devices. We observe that BPV (see Section 2.2) has been overlooked for various standard cryptographic suites. We harness BPV to speed-up operations involving a scalar multiplication with randomness in these cryptographic techniques.

- *Enabling BPV for Designated Public Keys:* Some of these integrated cryptographic techniques require an online scalar multiplication over a public key in the form of $\langle r, r \times \mathbf{U} \rangle$, which cannot be directly speed-up via BPV. However, we observe that it is possible to extend BPV to this setting, if the sender can store a table for each receiver public key $\langle \Gamma_i, \mathbf{U}_i \rangle_{i=1}^{r'}$. In many IoT applications, the number of receivers that an IoT device reports to is generally limited (one or at most a few cloud servers). Hence, we propose to apply BPV to this set of designated public keys, and we refer this strategy to as *Designated BPV (DBPV)*. Please note that *DBPV* might not be applicable if the number of receivers is large for the IoT device.

- *Preserving Security Features of Primitives due to Direct Integration:* All the improved proposed schemes perpetuate security properties of underlying primitives as optimization techniques are integrated directly, without any modification. Therefore, there is no need for separate security proofs of the proposed schemes. Thus, our optimizations can be integrated easily to the existing schemes.

- **Improving AQ and ECHMQV Key Exchange:**

Scheme I - Ephemeral AQ-BPV: Algorithm 1 depicts that E_a and E_b are calculated by EC scalar multiplications. Instead, we leverage BPV to minimize this overhead. Thus, in the offline phase, pre-computation steps of BPV are followed by both parties so that in the online phase E_a and E_b are calculated only with EC additions.

Scheme II - ECHMQV with AQ-BPV: ECHMQV protocol needs a prior ECDH key exchange, which requires certified public keys [37]. Instead, we make ECHMQV self-certified by adopting Fixed AQ protocol. Prior to online calculations, nodes A and B follow Fixed AQ protocol [5]. Thus, private and public key pair of nodes are $x_a = [H(ID_a, \mathbf{U_a}) \cdot b_a + d]$ where $\mathbf{U_a} = b_a \times \mathbf{G}$ and $x_b = [H(ID_b, \mathbf{U_b}) \cdot b_b + d]$, where $\mathbf{U_b} = b_b \times \mathbf{G}$. Furthermore, ECHMQV also receives benefits from BPV, especially in deriving ephemeral session keys. Note that the values $H(ID_b||\mathbf{U_b}) \times \mathbf{U_b} + \mathbf{D}$ and $(H(ID_a||\mathbf{U_a}) \times \mathbf{U_a} + \mathbf{D})$ can be calculated only once, prior to online communications. With all these optimizations combined, a total of four EC scalar multiplications can

Algorithm 3 ECHMQV WITH AQ-BPV

```
1: Node A: $(p_a, \mathbf{P}_a) \leftarrow BPV.Online(\mathit{\Gamma}_a)$
2: Node B: $(p_b, \mathbf{P}_b) \leftarrow BPV.Online(\mathit{\Gamma}_b)$
3: Node A and Node B exchange $\mathbf{P}_a$ and $\mathbf{P}_b$
4: $e_1 \leftarrow H(\mathbf{P}_a||\mathbf{U_b})$ and $e_2 = H(\mathbf{P}_b||\mathbf{U_a})$
5: $\sigma_A \leftarrow (p_a + e_1 \cdot x_a) \times (\mathbf{P}_b + e_2 \times H(ID_b||\mathbf{U_b}) \times \mathbf{U_b} + \mathbf{D})$
6: $\sigma_B \leftarrow (p_b + e_2 \cdot x_b) \times (\mathbf{P}_a + e_1 \times H(ID_a||\mathbf{U_a}) \times \mathbf{U_a} + \mathbf{D})$
7: $\mathbf{K}_{ab} = H(\sigma_A) = H(\sigma_B)$
```

be reduced to $(3 + 2k)$ $Eadd$ ($Eadd$ denotes EC additions, $k = 8$), which offers a significant performance gain.

• **Improving Integrated Schemes:**

Scheme III - ECIES with AQ-BPV: As in ECHMQV, we first integrate Fixed AQ into ECIES to achieve self-certified fixed ECDH keys. Therefore, $x_a = [H(ID_a, \mathbf{U_a}) \cdot b_a + d]$ and $x_b = [H(ID_b, \mathbf{U_b}) \cdot b_b + d]$, where $\mathbf{U_a} = b_a \times \mathbf{G}$ and $\mathbf{U_b} = b_b \times \mathbf{G}$. Moreover, the sender uses BPV to eliminate an online EC scalar multiplication. Finally, $H(ID_b||\mathbf{U_b}) \times \mathbf{U_b}$ is calculated only once at the offline phase.

Algorithm 4 ECIES WITH AQ-BPV

```
Sender:
1: $(p_a, \mathbf{P}_a) \leftarrow BPV.Online(\mathit{\Gamma}_a)$
2: $\mathbf{Z} \leftarrow p_a \times [H(ID_b||\mathbf{U_b}) \times \mathbf{U_b} + \mathbf{D}]$, where $\mathbf{Z} = (x_1, y_1)$
3: $k_e||k_m \leftarrow KDF(S||S_1)$, where $S_1$ is public (e.g., $ID_a$) and $S = x_1$
4: $c \leftarrow \mathcal{E}_{k_e}(m)$, $d \leftarrow MAC_{k_m}(c||S_2)$, where $S_2$ is public (e.g., $ID_b$)
5: Send $(\mathbf{P}_a, c, d)$ to the receiver
Receiver:
1: $\mathbf{Z} \leftarrow x_b \times \mathbf{P}_a$, where $\mathbf{Z} = (x_1, y_1)$
2: $k_e||k_m \leftarrow KDF(S||S_1)$, where $S = x_1$
3: If $d = MAC_{k_m}(c||S_2)$ then $m \leftarrow \mathcal{D}_{k_e}(c)$
```

ECIES can be further improved with DBPV as follows:

Scheme IV - ECIES with AQ-DBPV: In addition to computing $\mathbf{P}_a$ with BPV (Sender Step 1), we observe that the values for public key $\mathbf{Z}$ can also be pre-computed and stored in a similar way. That is, our pre-computation table also includes values for $(p_a, \mathbf{Z} = p_a \times [H(ID_b||\mathbf{U_b}) \times \mathbf{U_b} + \mathbf{D}])$. When sender needs to generate S, she just uses these pre-computed values to obtain Z with only k *Eadd* operations. Therefore, we denote these DBPV operations as $(p_a, \mathbf{P}_a, \mathbf{Z}) \leftarrow \mathit{DBPV.Online}(\Gamma_a)$. Notice that, after these improvements, there is no EC scalar multiplications but only $2k$ *Eadd* operations at the sender side.

Scheme V - Signcryption with AQ-DBPV: We notice that Signcryption is initiated by sender performing an *Emul* over the public key of the receiver (a DH key in base Signcryption [64]). This implies that Signcryption can also benefit from both AQ and DBPV optimizations. That is, we first make Signcryption self-certified, where the nodes follow fixed AQ protocol prior to online communication as, $x_a = [H(ID_a, \mathbf{U_a}) \cdot b_a + d]$ and $x_b = [H(ID_b, \mathbf{U_b}) \cdot b_b + d]$, where $\mathbf{U_a} = b_a \times \mathbf{G}$ and $\mathbf{U_b} = b_b \times \mathbf{G}$, respectively. Furthermore, as in ECIES, the sender performs $(p_a, \mathbf{P}_a, \mathbf{Z}) \leftarrow \mathit{DBPV.Online}(\Gamma_a)$.

• **Security of Proposed Schemes:** Our security depends on two well-known primitives, AQ and BPV (considering DBPV is just an extension of BPV and incorporates its security).

The security of BPV is well-analyzed and relies on the hardness of Hidden Subset Sum Problem [18]. Moreover, the security of BPV with an integration to Elliptic Curve Discrete Logarithm Problem (ECDLP) based protocols (e.g., ECDSA) has been

Algorithm 5 Signcryption with AQ-DBPV

```
Sender:
  1: $(p_a, \mathbf{P}_a, \mathbf{Z}) \leftarrow DBPV.Online(\Gamma_a)$, where $\mathbf{Z} = (x_1, y_1)$
  2: $k_e || k_m \leftarrow H(x_1)$
  3: $r \leftarrow H(k_m || m)$ and $s \leftarrow p_a \cdot (r + x_a)^{-1} \bmod q$
  4: $c \leftarrow \mathcal{E}_{k_e}(m)$, output $(c, r, s)$
Receiver:
  1: $\mathbf{Z} = (s \cdot x_b) \times [(H(ID_a || \mathbf{U_a}) \times \mathbf{U_a} + \mathbf{D}) + r \times \mathbf{G}]$
  2: $k_e || k_m \leftarrow H(x_1)$, where $\mathbf{Z} = (x_1, y_1)$
  3: $m \leftarrow \mathcal{D}_{k_e}(c)$, accepted if $H(k_m || m) = r$
```

investigated in [6]. Specifically, the BPV with ECDLP based signatures rely on Affine Hidden Subset Sum Problem. Given that our adoption of BPV into ECDLP-based key exchange protocols, integrated scheme, and Signcryption adhere these principles, our techniques preserve these security guarantees.

Rest is to show that self-certification does not impact the security of the proposed schemes. As stated by Bernstein in [12], the signature $s = y - H(m||R) \cdot r \bmod q$ in *Schnorr* is a linear combination of the permanent private key y and the ephemeral private key r, with coefficients 1 and $H(m||R)$, respectively. Therefore, it is possible to modify these coefficients by any function of m and R, which yields several variants of *Schnorr* signature. Such variants are also called as "Schnorr-like signatures" as discussed in [28, 12]. Although it is not discussed in the original AQ paper [5], it is depicted in Algorithm 1 that the private key assigned to nodes is in this form. Basically, in AQ scheme, the private keys are Schnorr-like signatures that are generated by the certification authority in off-line phase and are verified during the key establishment phase. Hence, the security of AQ scheme relies on the security of Schnorr-like

Table 2.2: Performance of existing and improved key exchange schemes on 8-bit ATmega 2560.

Protocol¶	**CPU Time**† **(s)**	**Code Size (Byte)**	**Bandwidth (Byte)**	**Cert. Overhead**
ECDH+ECDSA+ Certificate	3.24	34698	72	yes
AQ	2.10	33192	**24**	**no**
ECHMQV+Certificate	4.31	35788	72	yes
Our Proposed Improved Schemes with Optimization				
AQ-BPV	**1.19**	45712	**24**	**no**
ECHMQV with BPV	**3.45**	45872	72	yes
ECHMQV with AQ-BPV	**2.26**	45872	**24**	**no**

¶ All protocols are implemented as ephemeral key exchange schemes. All comparisons are made for the online phases of these schemes.
† CPU times are based on the first online phase of the protocols. After the first phase, where the public key should be verified, verification cost (1.19s) is removed, until public keys are renewed.

signatures, which is well-analyzed.

2.4 Performance Evaluation

Experimental Setup and Evaluation Metrics: We implemented our schemes and their counterparts on an 8-bit ATmega 2560 microcontroller. ATmega 2560 is a very lightweight device and used commonly in practice for IoT applications, especially in medical devices [55, 56], where there are critical time and energy constraints. AVR ATmega 2560 is an 8-bit microcontroller with 256 KB flash memory, 8KB SRAM and 4 KB EEPROM and its maximum clock frequency is 16MHz. During our experiments, ATmega 2560 was powered by a 2200 mAh power pack. This enabled us to use a DC power monitor/ammeter connected between the battery and processor to monitor the current drawn. Moreover, the experimental current results are compared with

Table 2.3: Performance of existing and improved integrated schemes on 8-bit ATmega 2560.

Protocol	**CPU Time**[†]**(s)**	**Code Size (Byte)**	**Bandwidth (Byte)**	**Cert. Overhead**
ECIES with ECDSA+Certificate	3.25	34876	96	yes
Signcryption with ECDSA+Certificate	2.48	36418	96	yes
Our Proposed Improved Schemes with Optimization				
ECIES with BPV	**2.38**	48274	96	yes
ECIES with DBPV	**1.51**	55004	96	yes
Signcryption with DBPV	**1.41**	49318	96	yes
ECIES with AQ-BPV	**1.19**	48274	**48**	**no**
ECIES with AQ-DBPV	**0.32**	55004	**48**	**no**
Signcryption with AQ-DBPV	**0.22**	49318	**48**	**no**

† CPU times are based on the first online phase of the protocols. After the verification of the certificate, the verification cost (1.19s) is removed, until public keys are renewed.

the datasheet of the processor[1]. All of the schemes are implemented using microECC library [39].

We selected our elliptic curve as the NIST-recommended secp192 [19] (security parameter $\kappa = 96$). Although there are more efficient curves such as Curve25519 [11] and FourQ [21], NIST curves are the most common ones which are deployed in practice due to their standardization. Moreover, Curve25519 and FourQ offer very fast elliptic curve additions, therefore, we believe, our improvements would be even more effective in these curves. However, in this thesis we are following the conservative approach which is not in our favor and use the NIST recommended curves to show our techniques can achieve these numbers even in the slower but standardized curves.

Our evaluation metrics include computation, code size (for ATmega 2560), com-

[1]http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf

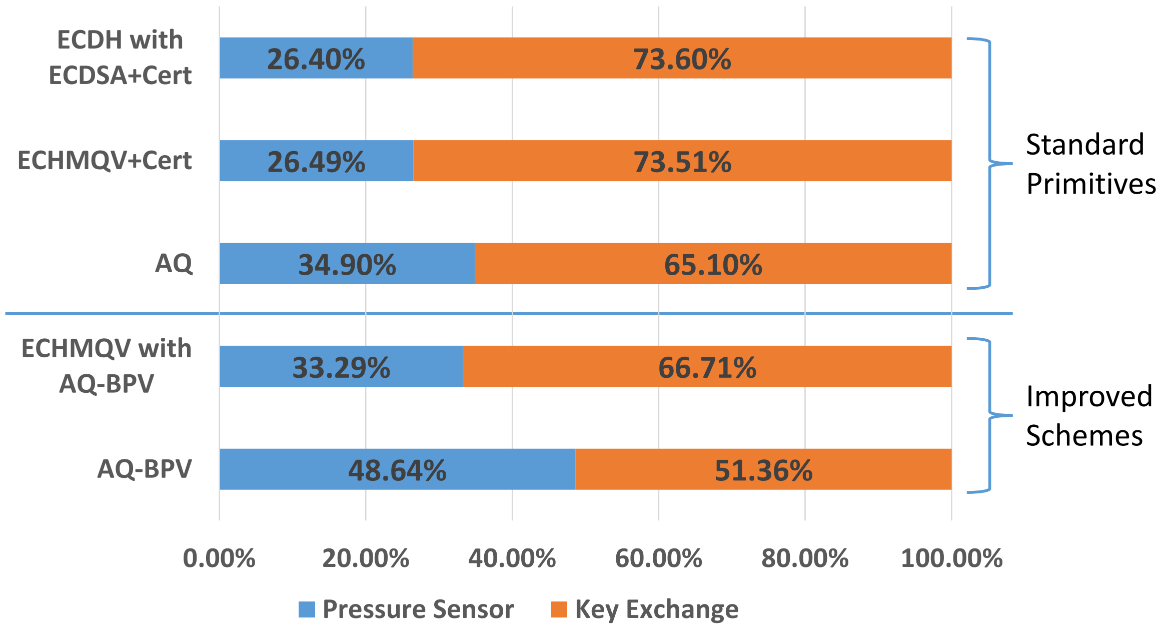

Figure 2.1: Energy comparison of key exchange protocols with IoT sensor (pressure) on 8-bit ATmega 2560

munication, memory overhead, and energy consumption. We measured the energy consumption with the formula $E = V \cdot I \cdot t$, where $V = 5$ Volts (required by ATmega2560), and t is the computation time (based on clock cycles) as in [6].

In our long-term experiments (to monitor the energy consumption), we focused on the dominative costs for all schemes. Therefore, we did not take the effect of certificate verification into consideration, as this will happen in the first online communication and may not be repeated until the receiver renews its public key. However, even if this cost was also considered, our advantages in terms of energy efficiency would increase.

Performance Evaluation and Comparison: Analytical comparison can be found in Appendix A. We give the experimental evaluation and comparison for key exchange and integrated protocols in Tables 2.2 and 2.3, respectively. Our exper-

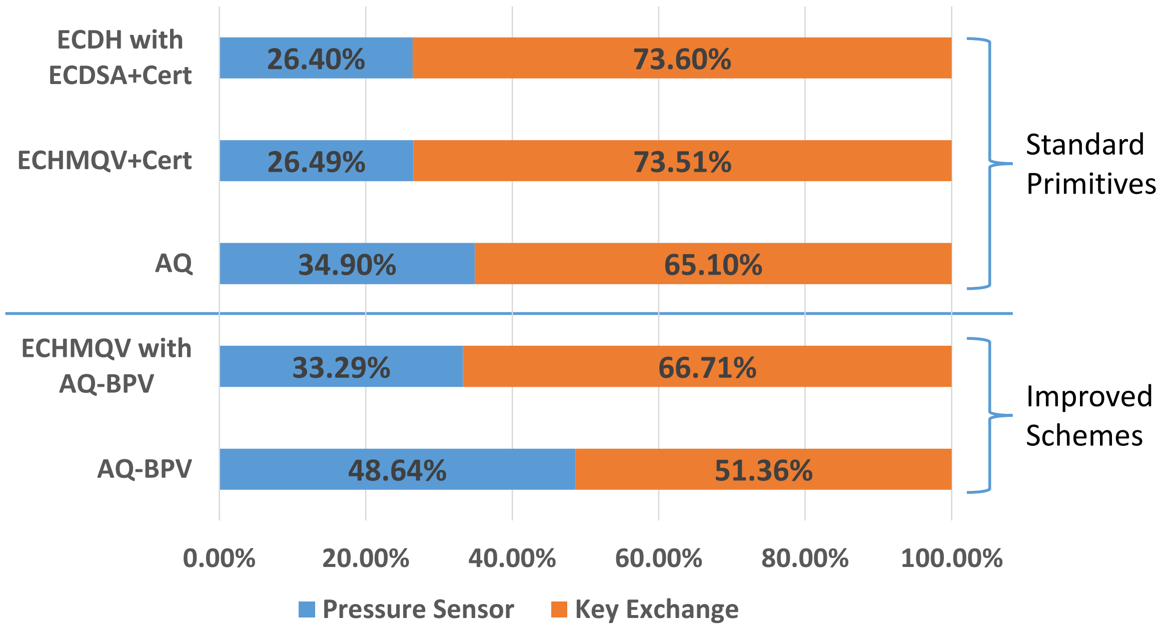

Figure 2.2: Energy comparison of integrated schemes with IoT sensor (pressure) on 8-bit ATmega 2560

iments confirmed significant improvements in terms of both CPU time and energy consumption. Moreover, besides saving more energy as compared to ECDH with certificates, they are more communication efficient, by reducing 48 Byte communication overhead. Our optimizations offer even better improvements for integrated protocols. ECIES and Signcryption with AQ-DBPV improve their base schemes for CPU time and energy efficiency by 6.44× and 5.86×, respectively.

In Figure 2.1 and Figure 2.2, we examined how much energy is required for cryptographic operations as compared to a BMP183 Pressure/Altitude Sensor[2] on ATmega 2560. To calculate the energy consumption of BMP183, we checked the datasheet and observed that the current drawn by the sensor is $5\mu A$ and it operates at 2.5V.

[2]`https://cdn-shop.adafruit.com/datasheets/1900_BMP183.pdf`

The sampling rate for this sensor is selected as 30 minutes, and the energy consumed by the sensor is calculated with the formula $E = V \cdot I \cdot t$. Additionally, ATmega 2560 consumes energy to read the data and also during the wait time. These energy consumptions are also taken into consideration. Results in Figure 2.1 and Figure 2.2 show that the cryptographic operations consume up to 73.6% of the battery. With our optimizations, this overhead is decreased to 51.36% and 16.38% for key exchange and integrated schemes, respectively.

Furthermore, we analyzed the time that ATmega 2560 can operate without a battery replacement/charge when both IoT sensor and cryptographic operations are used, and the sampling rate is 30 minutes. This analysis showed how much our optimizations on cryptographic operations affect the overall energy consumption of the IoT application. We used the data presented in Figure 2.1 and Figure 2.2 to analyze battery replacement time. If ECDH with ECDSA certificate or ECHMQV with ECDSA certificate were used, the battery would be drained in 50 days, this is increased to 92 days with AQ-BPV. Moreover, Signcryption with certificates and ECIES with certificates drain the battery in 88 and 67 days respectively. With our improved schemes, these numbers increase to 158, 148 days.

Chapter 3: Attacks and Fixes on Real-Time Signatures for Critical Cyber Infrastructures

3.1 Our Contributions

Our contributions are two-fold: (i) We show a security flaw in SCRA digital signature scheme in [61]. (ii) We then propose a new series of fast digital signatures that can address the performance and security issues of the previous constructions in [60, 61]. We further elaborate our contributions as follows.

Attack on SCRA Signature Scheme: SCRA signature generation aggregates signatures computed from only a small set of messages, and then releases both the aggregate signature and indexes of the corresponding messages. The exposure of the indexes and aggregate signature without a one-time masking leaks information about the private key. That is, we show that it is possible to recover a SCRA private key by observing only 8192 signatures with an overwhelming probability, for an instantiation.

Fast Authentication from Aggregate Signatures (FAAS): We develop a new signature scheme that we refer to as *Fast Authentication with Aggregate Signatures* (`FAAS`). `FAAS` can be instantiated from any single-signer (k-element extraction secure [16]) aggregate signature. We propose two efficient instantiations of `FAAS`: (i) An instantiation with *Condensed-RSA* [41] called as `FAAS-C-RSA`, and (ii) an instantiation with *pqNTRUsign* [34] called as `FAAS-NTRU`. The desirable properties of `FAAS` instantiations are as follows.

- *Fast Signing*: `FAAS` instantiations offer significant speed improvements over their base schemes in terms of signature generation, since they do not require expensive operations (e.g., exponentiation, Gaussian sampling). More specifically, `FAAS-C-RSA` improves its base scheme's signature generation by $42\times$, while this improvement is $30\times$ for `FAAS-NTRU`. Moreover, unlike its base scheme, `FAAS-NTRU` does not require any rejection sampling, and therefore offers a constant signature generation time.

- *Low End-to-end Delay*: We instantiate `FAAS` with verification-efficient digital signature schemes to complement the benefit of highly efficient signature generation. This strategy enables a low end-to-end cryptographic delay for `FAAS` instantiations. For instance, `FAAS-C-RSA` and `FAAS-NTRU` improve the end-to-end delay of their base schemes by $32\times$ and $17\times$, respectively.

- *Improved Side-Channel Resiliency*: Side-channel attacks pose a critical threat to real-life implementations of cryptographic schemes. For instance, several side-channel attacks have been shown (e.g., [30, 26]) against lattice-based digital signature schemes that rely on Gaussian sampling (e.g., [23]). Moreover, the use of weak pseudorandom number generators (PRNGs) in digital signatures are also known to cause security vulnerabilities [45, 35]. `FAAS` instantiations offer features that can mitigate some of these attacks: (i) `FAAS` instantiations do not require operations such as exponentiation or Gaussian sampling in their online signature generation algorithm. Therefore, they have an improved resiliency against side-channel attacks that target these operations. (ii) `FAAS` instantiations do not generate any new randomness at their signature generation (as in [14]), and therefore can avoid the problems that stem from weak PRNGs.

- *Post-Quantum Promise*: `FAAS-NTRU` inherits the post-quantum promises of its base scheme, and therefore, is potentially a suitable choice for applications that require long-term security with high signing efficiency and low end-to-end delay.

- *Structure-free Signing with Constant-size Secret Key*: Unlike RA [60], which assumes a pre-defined (fixed length) format in messages and relies on one-time masking signatures (that results in linear storage (or computation) overhead), `FAAS` instantiations do not require a format on messages, and have a constant-size private key. Therefore, `FAAS` instantiations can potentially meet the demands of various delay-aware applications.

Limitations: The main limitation of `FAAS` instantiations is the increased private key size due to the storage of pre-computed signatures. For example, `FAAS-C-RSA` and `FAAS-NTRU` require 193 KB and 511 KB private keys, respectively, with $\kappa = 128$-bit security (see Table 3.4 in Section 3.6 for details). This increased private key size; however, translates into 30× and 42× faster signing with an improved side-channel resiliency, for `FAAS-C-RSA` and `FAAS-NTRU`, respectively. This is expected to be a highly desirable feature for some applications that demand delay-aware authentication (e.g., energy delivery systems, vehicular networks) and the signer can store such private keys (e.g., command center, car). On the other hand, `FAAS` instantiations might not be ideal for highly memory limited signers (e.g., a medical implantable).

Potential Use-cases: Some use-cases for `FAAS` schemes, including but not limited to, could be listed as follows.

(i) Smart-Grids: Phasor Measurement Units (PMUs) enable high rate data transfer, which demands high-speed authentication [57]. Moreover, considering that PMUs are usually low-end devices (e.g., 32-bit ARM microcontrollers), delay requirements of smart-grid systems may not be achieved with standard techniques. `FAAS` schemes seem to be a suitable alternative to meet real-time authentication need of PMUs. `FAAS` schemes offer significant improvements on the signature generation time of their base schemes, and therefore might be an ideal choice for smart-grid systems. `FAAS` can meet stringent 4 to 20 ms end-to-end delay requirement [2] of smart-grid systems. Notice that, in practice, PMUs rely on capable but still low-end hardware (e.g., simulated with ARM Cortex A-9 processors equipped with 1GB storage and 512 MB DDR3 memory in [57]). We expect that such processors can readily store the large private keys of `FAAS` schemes.

(ii) Vehicular and Drone Networks: Another use-case is high-speed signature generation in vehicular networks. For instance, according to relevant standards [33, 3], a vehicle may need to generate up to 1000 messages per second. ECDSA has been considered as the choice of a digital signature in these standards; however, the shortcomings of ECDSA at this front have been shown [40, 63]. Given high throughput advantages, along with its low end-to-end delay, `FAAS` schemes might be a suitable alternative to ECDSA. We also observe that `FAAS` schemes can be useful to secure the

communication channel and telemetry for commercial air drone networks. Recently, the need for efficient cryptographic techniques for commercial air drones started to receive attention from the research community [59, 50]. `FAAS` schemes can be used for the broadcast authentication of telemetry data from an off the shelf drone. Such high-speed authentication techniques can be useful for commercial applications like Amazon Air[1].

Remark that, in all aforementioned use-cases, the sender can potentially store a larger private key.

3.2 Preliminaries

Notation. $|a|$ denotes the bit length of variable a. $a \xleftarrow{\$} S$ denotes that a is selected from set S at random. In $x||y$, $||$ denotes the concatenation of bit strings of x and y. We represent vectors as bold letters $\mathbf{a}$, while scalars are represented as non-bold letters i.e., a. $||\mathbf{a}||_2$ and $||\mathbf{a}||_\infty$ denote the Euclidean norm and infinity norm of vector ν, respectively. We define hash functions $H_0 : \{0,1\}^* \rightarrow \{0,1\}^{l_0}$, $H_1 : \{0,1\}^{l_1} \rightarrow \{0,1\}^*$ and $H_2 : \{0,1\}^* \rightarrow \{0,1\}^{l_2}$ for some integers l_0, l_1 and l_2. $\mathcal{A}^{\mathcal{O}_1,\ldots,\mathcal{O}_i}$ denotes that algorithm $\mathcal{A}$ is provided with access to oracles $\mathcal{O}_1, \ldots, \mathcal{O}_i$.

Definition 1 *A single-signer aggregate signature ASig is defined as follows:*

- $(sk, PK) \leftarrow$ `Asig.Kg`(1^κ): *Given the security parameter* 1^κ *as the input, the key generation algorithm returns a private/public key pair* (sk, PK) *as the output.*

[1]`https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011`

- $\gamma \leftarrow \texttt{Asig.Sig}(m, sk)$: *Given a message* $m \in \{0,1\}^*$ *and private key* sk *as the input, this algorithm returns a signature* γ *of the message under* sk *as the output.*

- $s \leftarrow \texttt{Asig.Agg}(\gamma_1, \ldots, \gamma_L)$: *Given a set of signatures* $(\gamma_1, \ldots, \gamma_L)$ *as the input, this algorithm returns a single-compact signature* s *as the output.*

- $\{0,1\} \leftarrow \texttt{Asig.Ver}(\overrightarrow{m}, s, PK)$: *Given messages* $\overrightarrow{m} = (m_1, \ldots, m_L)$*, aggregate signature* s *and* PK *as the input, this algorithm returns a bit:* 1 *means* valid *and* 0 *means* invalid.

Definition 2 *`Agg` function that is used to aggregate multiple messages to a single message is defined as follows:*

- $(m) \leftarrow \texttt{Agg}(m_1, \ldots, m_L)$: *Given a set of messages* $(m_1, \ldots, m_L)$ *as the input, `Agg` function returns a single message* m *as the output.*

`Agg` function is also a part of the `Asig.Ver` algorithm that allows the batch verification of multiple messages. This function can be instantiated as multiplication over mod N in RSA [49] or vector addition in pqNTRUsign [34].

`FAAS` is proven to be *Aggregate Existential Unforgeability under Chosen Message Attack* (*A-EU-CMA*) [16] based on the experiment defined in Definition 3. In this experiment, the adversary $\mathcal{A}$ is provided with access to the following oracles:

- *Setup* $RO(\cdot)$ *Oracle*: $\mathcal{A}$ is given access to a *random oracle* $RO(\cdot)$ [10], from which it can request the hash of any input x for up to q_H messages. The function *H-Sim* is used to handle $RO(\cdot)$ queries. That is, the cryptographic hash function H is modeled as a random oracle via *H-Sim* as follows.

  – $h \leftarrow H\text{-}Sim(x, L_H, i)$: If $x \in L_H$ then $H\text{-}Sim$ returns the corresponding value $h \leftarrow L_H(x)$. Otherwise, it returns $h \overset{\$}{\leftarrow} \{0,1\}^{l_i}$ where $i \in \{0,1,2\}$ as the answer, inserts (x, h) into L_H, respectively.

- $\sigma \leftarrow \mathtt{SigA}_{sk}(\overrightarrow{m})$: $\mathcal{A}$ is provided with the *aggregate signing oracle* $\mathtt{SigA}_{sk}(.)$. For each aggregate signature query i, $\mathcal{A}$ queries $\mathtt{SigA}_{sk}(\cdot)$ with a batch message $\overrightarrow{m} = (m_1, \ldots, m_j)$ for some integer j, and will be provided with the corresponding aggregate signature σ computed under sk. This oracle keeps track of all the submitted queries by keeping a list L_m, which is initially empty. $\mathcal{A}$ can query this oracle to up to q_S messages. $\mathtt{SigA}_{sk}(.)$ is defined in the *A-EU-CMA* experiment (as in Definition 3).

Definition 3 *Aggregate Existential Unforgeability under Chosen Message Attack (A-EU-CMA) for a single user aggregate signature is as follows.*

```
$Exp^{A\text{-}EU\text{-}CMA}_{\mathtt{Asig},\mathcal{A}}(1^{\kappa})$ :
  $L_m \leftarrow \emptyset$
  $(sk, PK) \leftarrow \mathtt{Asig.Kg}(1^{\kappa})$
  $(\overrightarrow{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathtt{SigA}_{sk}(\cdot), RO(\cdot)}(PK)$
```

```
$\mathtt{SigA}_{sk}(\overrightarrow{m})$
    $\gamma_i \leftarrow \mathtt{Asig.Sig}(m_i, sk)$ for $i = 1, \ldots, j$
    $\sigma \leftarrow \mathtt{Asig.Agg}(\gamma_1, \ldots, \gamma_j)$
    $L_m \leftarrow L_m \cup \overrightarrow{m}$
```

We say $\mathcal{A}$ wins in time t, and after q_S and q_h queries if $((\mathtt{Asig.Ver}(\overrightarrow{m}^*, \sigma^*, PK) \wedge (\overrightarrow{m}^* \cap L_m = \emptyset))$. *The A-EU-CMA advantage of* $\mathcal{A}$ is defined as $Adv^{A\text{-}EU\text{-}CMA}_{\mathtt{Asig},\mathcal{A}}(t, q_S, q_h) = \Pr[Exp^{A\text{-}EU\text{-}CMA}_{\mathtt{Asig},\mathcal{A}} = 1]$.

`FAAS` requires that the underlying aggregate signature achieves *k-element Aggre-*

gate Extraction (AE) property [16, 20], which is defined in the following.

Definition 4 *For a given aggregate signature $s \leftarrow \texttt{SigA}_{sk}(\overrightarrow{m})$ computed on k individual data items $\overrightarrow{m} = (m_1, \ldots, m_k)$, it is difficult to extract the individual signatures $(\gamma_1, \ldots, \gamma_k)$ of $(m_1, \ldots, m_k)$ provided that only s is known to the extractor.*

Initially, Boneh et al. [16] assumed that it is a hard problem to extract individual BGLS signatures [15] given an aggregate BGLS signature, which was then proven to hold in [20] under the Computational Diffie-Hellmann assumption. We note that C-RSA [43] and pqNTRUsign [34], which are used in `FAAS` instantiations, achieve this property (see B for a discussion on pqNTRUsign).

3.3 Attack on SCRA Signature Scheme

We identified a security flaw in the signature scheme called *Structure-free and Compact Rapid Authentication (SCRA)* [61]. SCRA was proposed as a generic scheme that transforms a single-signer aggregate signature scheme into a fast signature scheme. As shown in Algorithm 6, in `SCRA.Kg` Steps 2-3, a set of signatures are pre-computed for each b-bit structure of L fields of the hash output (where $b \cdot L = l_0$). These signatures are stored in a pre-computed table comprised of $2^b \cdot L$ signatures, which is the private key of SCRA (see Γ in `SCRA.Kg`). In the signature generation algorithm, the message is hashed with a randomness, and for each of L fields of the hash output, their corresponding pre-computed signatures are aggregated. The randomness is sent along with the aggregated signature to enable signature verification. Thus, in Steps

Algorithm 6 Structure-free and Compact Real-time Authentication (*SCRA*)

```
$\underline{(sk, PK) \leftarrow \texttt{SCRA.Kg}(1^\kappa)}$:
1: $(sk', PK') \leftarrow \texttt{ASig.Kg}(1^\kappa)$, $P \leftarrow \{0,1\}^{l_0}$
2: Select integers $(b, L)$ such that $b \cdot L = l_0$
3: $\tilde{m}_{i,j} \leftarrow i||j||P$, $\gamma_{i,j} \leftarrow \texttt{ASig.Sig}(\tilde{m}_{i,j}, sk')$, $i = 1, \ldots, L$ and $j = 0, \ldots, 2^b - 1$
4: $sk \leftarrow (sk', \Gamma)$ and $PK \leftarrow (PK', P)$, where $\Gamma \leftarrow \{\tilde{m}_{i,j}, \gamma_{i,j}\}_{i=1,j=0}^{L,2^b-1}$

$\underline{\sigma \leftarrow \texttt{SCRA.Sig}(m, sk)}$:
1: $(M_1^*, \ldots M_L^*) \leftarrow H_0(m||r)$ where $r \in \{0,1\}^\kappa$ and $M_i^* \in [0, 2^b - 1]$, $i = 1, \ldots, L$.
2: $m_i' \leftarrow i||M_i^*||P$, and fetch corresponding signature $\gamma_i'$ of $m_i'$ from table $\Gamma$, $i = 1, \ldots, L$
3: $s \leftarrow \texttt{ASig.Agg}(\gamma_1', \ldots \gamma_L')$ and $\sigma \leftarrow (r, s)$

$\underline{\{0,1\} \leftarrow \texttt{SCRA.Ver}(m, \sigma, PK)}$:
1: $(M_1^*, \ldots, M_L^*) \leftarrow H_0(m||r)$
2: $m_i' \leftarrow i||M_i^*||P$, $i = 1, \ldots, L$
3: $\{0,1\} \leftarrow \texttt{ASig.Ver}(\langle m_1', \ldots, m_L' \rangle, s, PK')$
```

1-2 in `SCRA.Ver`, individual messages are recovered and their verification is performed in Step 3.

We noticed that for signature verification, SCRA allows the verifier to know which message sub-fields (i.e., indexes) are used during signature generation (i.e., aggregation). Therefore, an adversary $\mathcal{A}$ can keep track of the indexes of individual signatures that are used to compute the aggregate signatures. This can lead to an information leakage about the private keys on generic SCRA design, since the adversary learns which private key components are aggregated.

In the case of *pqNTRUsign instantiation*, after observing a small-constant number of aggregate signatures, $\mathcal{A}$ can re-construct either the whole or a part of the pre-computation table (private key) by solving a system of linear equations. The number of signatures necessary for a successful attack to recover the whole table depends on the signatures observed by $\mathcal{A}$. $\mathcal{A}$ needs to solve a set equations of L variables, with $2^b \cdot L$ unknowns, as can be seen in $s = \texttt{ASig.Agg}(\gamma'_1, \ldots, \gamma'_L)$. $\mathcal{A}$ can construct equations based on the indexes of the specific signatures. To recover all the private keys, the signatures observed by $\mathcal{A}$ should contain all the private key components. Since in each signature generation, one signature is selected from each L fields containing 2^b signatures, the probability that one signature is not selected is $(2^b - 1/2^b)^{q_S}$, where q_S denotes the number of signatures observed by $\mathcal{A}$. Considering that there are $2^b \cdot L$ different pre-computed signatures in SCRA private key, the probability that $\mathcal{A}$ derives all of the secret key components ($\Pr[\mathit{Win}_{\mathcal{A}}]$) is:

$$\Pr[\mathit{Win}_{\mathcal{A}}] = 1 - (2^b \cdot L \cdot (\frac{2^b - 1}{2^b})^{q_S}), \text{ where } q_S \geq 2^b \cdot L \tag{3.1}$$

Therefore, with an overwhelming probability, all of the SCRA - pqNTRUsign instantiation private key can be recovered by observing $2^b \cdot L$ signatures. Considering that suggested parameters for (b, l) are $(8, 32)$ [61], when $q_S = 8192$, $\mathcal{A}$ can recover all the private key components with probability ≈ 1. Although SCRA can be instantiated with different (b, l) parameters, since the storage overhead introduced is equal to $2^b \cdot L$, this number cannot be large enough to prevent our attack.

In the following, we show that it is possible to design a signer-efficient signature scheme from aggregate signatures while addressing the security issues of SCRA.

3.4 The Proposed Scheme

Pre-computed One-time Masking for Aggregate Signatures: We note that the security flaw in SCRA stems from the lack of one-time masking in signature generation. Since the indexes (message encoding) must be released for verification, one can recover the private key of SCRA. *To overcome this in an efficient manner, we propose a one-time masking strategy based on pre-computed signature components that can be applied to any aggregate signature scheme.* Our idea is to (deterministically) generate random message components and their corresponding signatures at the key generation phase and then aggregate a subset of them to generate a random message-signature pair. Since aggregation is usually faster than signature generation (e.g., in RSA [49], pqNTRUsign [34]), this allows us to efficiently generate a random message-signature pair. Then, this pair can be used to hide the SCRA private keys at signature generation phase, by only aggregating this with the SCRA signature (see Figure 3.1). Moreover, the aggregated message should be released with the signature, however since one cannot derive neither the individual message components nor the selected indexes (signer does not release them), our attack cannot be applied to this technique.
Main Idea: We capitalize on the observation in [60, 61] that the signature aggregation of some signature schemes is significantly faster than that of their signature generation. This allows the pre-computation of a set of signatures at the key generation (offline), some subset of which will later be efficiently aggregated during the signature generation (online) to sign a message depending on its encodings. However, `FAAS` differs from previous constructions in the way that messages and randomness

are encoded and computed: (i) In FAAS, the message is encoded as L b-bit structures, however unlike SCRA, we do not sample different structures from “fields” and rely on L-out-of-2^b different combinations (as in [17]). This strategy significantly reduces the private key size of FAAS schemes when compared to SCRA. (ii) FAAS hides individual aggregate signatures that encode a message with a random masking signature with the described strategy. Unlike RA [60], this approach eliminates the requirement of storing and re-generating linear number (with respect to the number of messages to be signed) of pre-computed online/offline signatures. Similarly, this permits FAAS to hide the pre-computed signatures that encode the messages, and therefore avoids the security vulnerability of SCRA [61] (recall Section 3.3).

All these strategies enable FAAS to transform any single-signer (k-element extraction secure as in Definition 4) aggregate signature scheme into a signer-efficient signature by only storing a small-constant size pre-computed signature table at the signer's side, without inheriting the performance or security problems of its previous counterparts in [60, 61].

3.4.1 Generic FAAS Design

Generic FAAS is presented in Algorithm 7 and Figure 3.1, and is further elaborated as follows.

Key Generation: In Step 1-2, we first generate parameters (k, t) and (b, L), which are used to determine a subset for the masking term and encode messages, respectively. We then create the private/public key pair of the underlying aggregate sig-

Algorithm 7 Generic FAAS Scheme

```
$(sk, PK) \leftarrow \texttt{FAAS.Kg}(1^\kappa)$:
1: Select integers $(k, t)$ such that $k \cdot |t| = l_2$ and $(b, L)$ such that $b \cdot L = l_0$
2: $(sk', PK') \leftarrow \texttt{ASig.Kg}(1^\kappa)$ and $z \leftarrow \{0,1\}^\kappa$
3: $u_i \leftarrow H_1(i||z)$ and $\beta_i \leftarrow \texttt{ASig.Sig}(u_i, sk')$ for $i = 0, \ldots, t-1$
4: Set pre-computed signature table $B \leftarrow \{\beta_i\}_{i=0}^{t-1}$
5: $\gamma_i \leftarrow \texttt{ASig.Sig}(i, sk')$ for $i = 0, \ldots, 2^b - 1$
6: Set pre-computed signature table $\Gamma \leftarrow \{\gamma_i\}_{i=0}^{2^b-1}$
7: $sk \leftarrow (z, B, \Gamma)$ and $PK \leftarrow PK'$

$\sigma \leftarrow \texttt{FAAS.Sig}(m, sk)$:
1: $(j_1, \ldots, j_k) \leftarrow H_2(m||z)$, where each $\{j_i\}_{i=1}^k$ is interpreted as a $|t|$-bit integer.
2: $u_{j_i} \leftarrow H_1(j_i||z)$ for $i = 1, \ldots, k$
3: $\sigma_U \leftarrow \texttt{ASig.Agg}(\beta_{j_1}, \ldots, \beta_{j_k})$
4: $u \leftarrow Agg(u_{j_1}, \ldots, u_{j_k})$
5: $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||u)$, where each $\{j_i\}_{i=1}^L$ is interpreted as a $b$-bit integer.
6: $s \leftarrow \texttt{ASig.Agg}(\sigma_U, \gamma_{j_1^*}, \ldots, \gamma_{j_L^*})$ and set $\sigma \leftarrow (u, s)$

$\{0,1\} \leftarrow \texttt{FAAS.Ver}(m, \sigma, PK)$:
1: $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||u)$
2: $\{0,1\} \leftarrow \texttt{ASig.Ver}(\langle u, j_1^*, \ldots, j_L^* \rangle, s, PK')$
```

nature and a random seed value z, which are used to generate two pre-computed signature tables: (i) In Step 3-4, we deterministically derive t random numbers with a keyed hash and compute their corresponding individual signatures to be stored in table B. A subset of these random numbers and table B will be aggregated (FAAS.Sig Step 2-3) and used to mask the private key components (FAAS.Sig Step 5). (ii) In Step 5-6, we generate 2^b signatures, from which L of them will be selected to encode the message in signature generation (FAAS.Sig Step 4-5). Finally, pre-computed tables and seed are stored as FAAS private key, while the public key of the

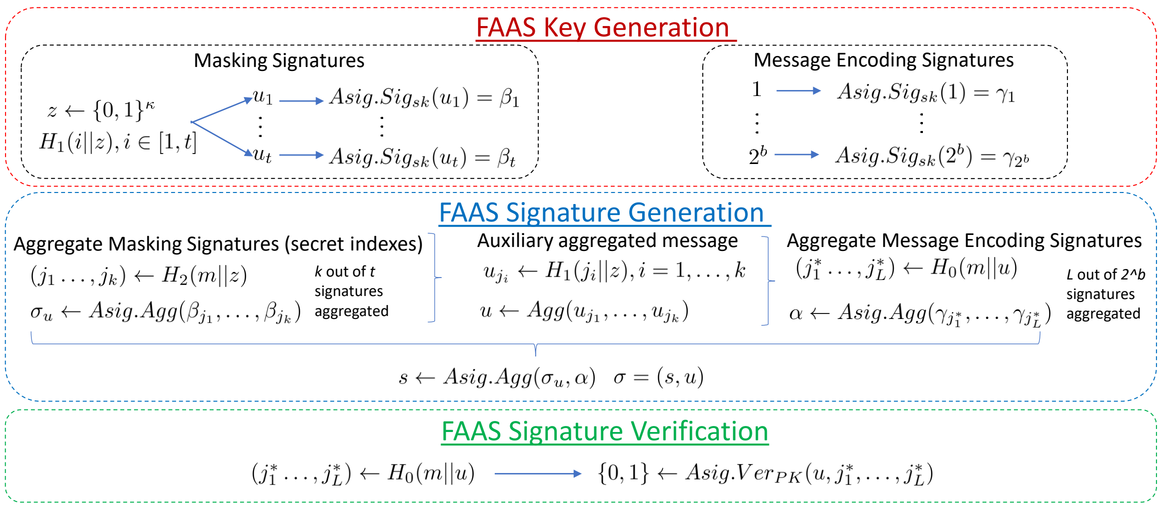
FAAS Key Generation
Masking Signatures
$z \leftarrow \{0,1\}^\kappa$
$H_1(i||z), i \in [1,t]$
$u_1 \rightarrow Asig.Sig_{sk}(u_1) = \beta_1$
$u_t \rightarrow Asig.Sig_{sk}(u_t) = \beta_t$
Message Encoding Signatures
$1 \rightarrow Asig.Sig_{sk}(1) = \gamma_1$
$2^b \rightarrow Asig.Sig_{sk}(2^b) = \gamma_{2^b}$
FAAS Signature Generation
Aggregate Masking Signatures (secret indexes)
$(j_1 \dots, j_k) \leftarrow H_2(m||z)$
$\sigma_u \leftarrow Asig.Agg(\beta_{j_1}, \dots, \beta_{j_k})$
k out of t signatures aggregated
Auxiliary aggregated message
$u_{j_i} \leftarrow H_1(j_i||z), i = 1, \dots, k$
$u \leftarrow Agg(u_{j_1}, \dots, u_{j_k})$
Aggregate Message Encoding Signatures
$(j_1^* \dots, j_L^*) \leftarrow H_0(m||u)$
$\alpha \leftarrow Asig.Agg(\gamma_{j_1^*}, \dots, \gamma_{j_L^*})$
L out of 2^b signatures aggregated
$s \leftarrow Asig.Agg(\sigma_u, \alpha) \quad \sigma = (s, u)$
FAAS Signature Verification
$(j_1^* \dots, j_L^*) \leftarrow H_0(m||u)$
$\{0,1\} \leftarrow Asig.Ver_{PK}(u, j_1^*, \dots, j_L^*)$

Figure 3.1: High-level description of FAAS algorithms.

underlying aggregate signature scheme is used as FAAS public key.

Signature Generation: In Step 1-2, we derive secret indexes $(j_1, \dots, j_k)$ from the message m and compute their corresponding random numbers $(u_{j_1}, \dots, u_{j_k})$ via a keyed hash. In Step 3-4, we first set u as the aggregation of the random numbers $(u_{j_1}, \dots, u_{j_k})$. Then, we aggregate their corresponding signatures selected from table B, as σ_U. In Step 5-6, we first encode the message and u together with indexes $(j_1^*, \dots, j_L^*)$, and then mask the aggregation of $(\gamma_{j_1^*}, \dots, \gamma_{j_1^*})$ with σ_U as $s \leftarrow \texttt{ASig.Agg}(\sigma_U, \gamma_{j_1^*}, \dots, \gamma_{j_1^*})$. We set FAAS signature as $\sigma = (u, s)$, where the aggregated randomness u enables the verification of the message m with s at the verifier side. Remark that it is essential to keep individual random numbers and their indexes as secrets by aggregating them. We do this with an aggregation function $u \leftarrow Agg(u_{j_1^*}, \dots, u_{j_k^*})$ as in Step 4. The aggregation function $Agg(.)$ is instantiated as modular multiplication and vector addition in Condensed RSA (C-RSA) [43] and pqNTRUSign [34], in Section 3.4.2. respectively.

Signature Verification: The signature verification simply checks whether the messages u and indexes $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||u)$ are verified with s under PK'.

Correctness of FAAS Design: `FAAS` signature is composed of two components, a random message u and an aggregated signature s. In essence, s is generated by aggregating $\beta_{j_1}, \ldots, \beta_{j_k}$ and $\gamma_{j_1^*}, \ldots, \gamma_{j_L^*}$ (see Steps 3 and 6 in `FAAS.Sig`), where indexes of β's are kept secret and indexes of γ's are released. Since, indexes of β's are secret, for the verification to work, the messages that corresponds to the selected signatures are also aggregated and released as u (see Step 4 in `FAAS.Sig`). This message aggregation is instantiated as multiplication over modulo N and vector addition in `FAAS-C-RSA` and `FAAS-NTRU`, respectively. Therefore, verifier first obtains the indexes (i.e., the messages, see Step 5 in `FAAS.Kg`) and then performs a batch verification. Considering that the messages involved in the batch verification are $\langle u, j_1^*, \ldots, j_L^* \rangle$, these correspond to the all signatures that are aggregated to obtain s. Therefore, our `FAAS` design is correct.

3.4.2 FAAS Instantiations

`FAAS` can be instantiated with any single-signer (k-element extraction secure) aggregate signature scheme. `FAAS` enables a highly fast signing for aggregate signature schemes with the cost of a larger private key size. Therefore, `FAAS` is especially beneficial for aggregate signature schemes with costly signing and efficient verification algorithms. This significantly improved signing efficiency, combined with already fast

Algorithm 8 Instantiation of `FAAS` with *C-RSA*

```
$(sk, PK) \leftarrow$ FAAS-C-RSA.Kg$(1^\kappa)$:
1: Generate two large primes $(p, q)$ and $n \leftarrow p \cdot q$. Compute $(e, d)$ such that $e \cdot d = \phi(n)$, where $\phi(n) \leftarrow (p-1)(q-1)$
2: $sk' \leftarrow (n, d)$, $PK' \leftarrow (n, e)$ and $z \leftarrow \{0,1\}^\kappa$
3: Select integers $(k, t, b, L)$ as in generic FAAS.Kg Step 1
4: $u_i \leftarrow H_1(i||z)$ and $\beta_i \leftarrow {u_i}^d \bmod n$, where $|u_i| = |n|$ for $i = 0, \ldots, t-1$
5: Set pre-computed signature table $B \leftarrow \{\beta_i\}_{i=0}^{t-1}$
6: $\gamma_i \leftarrow H_F(i)^d \bmod n$, for $i = 0, \ldots, 2^b - 1$, where $H_F : \{0,1\}^* \rightarrow Z_n^*$
7: Set pre-computed signature table $\Gamma \leftarrow \{\gamma_i\}_{i=0}^{2^b-1}$
8: $sk \leftarrow (z, B, \Gamma)$ and $PK \leftarrow PK'$
```

```
$\sigma \leftarrow$ FAAS-C-RSA.Sig$(m, sk)$:
1: $(j_1, \ldots, j_k) \leftarrow H_2(m||z)$, where each $\{j_i\}_{i=1}^k$ is interpreted as a $|t|$-bit integer.
2: $u_{j_i} \leftarrow H_1(j_i||z)$ for $i = 1, \ldots, k$
3: $u \leftarrow \prod_{i=1}^k u_{j_i} \bmod n$ and $\sigma_U \leftarrow \prod_{i=1}^k \beta_{j_i} \bmod n$
4: $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||u)$, where each $\{j_i\}_{i=1}^L$ is interpreted as a $b$-bit integer.
5: $s \leftarrow \sigma_U \cdot \prod_{i=1}^L \gamma_{j_i^*} \bmod n$ and set $\sigma \leftarrow (u, s)$
```

```
$\{0,1\} \leftarrow$ FAAS-C-RSA.Ver$(m, \sigma, PK)$:
1: $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||u)$
2: if $s^e = u \cdot \prod_{i=1}^L H_F(j_i^*) \bmod n$ then return 1, else return 0
3: end if
```

verification, enable `FAAS` to achieve a low end-to-end delay. In this line, we propose two efficient instantiations of `FAAS` as below.

`FAAS-C-RSA` : We instantiate `FAAS` with C-RSA, that is proven to be secure under the RSA assumption in the random oracle model [9]. Note that the signature generation of C-RSA is expensive since it requires an exponentiation over a large modulus, whereas its verification only requires an exponentiation over a small modulus (e.g., 65537). Therefore, `FAAS-C-RSA` has significant improvements over C-RSA in terms of signature generation and end-to-end delay. The detailed description of `FAAS-C-RSA` is given in Algorithm 8.

`FAAS-NTRU`: Lattice-based signature schemes provide a viable post-quantum security promise [24]. We identified only a few lattice-based signature schemes that are proven to have secure aggregation [34, 25]. Among these schemes, we noticed that pqNTRUsign [34] offers fast verification with a slow signature generation that requires Gaussian sampling. Thus, `FAAS-NTRU` vastly improves the signing efficiency of pqNTRUsign as well as achieving an improved side-channel resiliency, thanks to the elimination of Gaussian sampling from the signature generation algorithm. The detailed description of `FAAS-NTRU` is presented in Algorithm 10, that refers to pqNTRUsign signature generation algorithm defined in Algorithm 9, to refrain from repetitions in the algorithm description. Notice that expensive calculations such as Gaussian sampling and polynomial multiplication are done in the key generation algorithm (once and offline). At the signing phase, only polynomial additions are performed.

As depicted in Algorithm 10, we work over a polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^N + 1)$

Algorithm 9 pqNTRUsign Signature Generation [34]

```
    $\underline{(\mathbf{v}') \leftarrow \texttt{pqNTRUsign.Sig}(m, sk', \mathbf{h})}$:
 1: Compute $(\mathbf{u_p}, \mathbf{v_p}) = H_N(m||\mathbf{h})$
 2: Sample $\mathbf{r} \leftarrow \chi_{\sigma}^{N}$ and $b \xleftarrow{\$} \{0,1\}$
 3: Compute $\mathbf{u_1} \leftarrow \mathbf{pr} + \mathbf{u_p}$, $\mathbf{v_1} \leftarrow \mathbf{u_1 h} \mod q$ and $\mathbf{a} \leftarrow (\mathbf{v_p} - \mathbf{v_1})/\mathbf{g} \mod p$
 4: if $||\mathbf{af}||_2 > \nu_s$ or $||\mathbf{ag}||_\infty > \nu_t$ then go to Step 2
 5: end if
 6: $\mathbf{v}' = \mathbf{v}_1 + (-1)^b \mathbf{ag}$
 7: if $||\mathbf{v}'||_\infty > q/2 - B_t$ then go to Step 2
 8: end if
```

Algorithm 10 `FAAS` pqNTRUsign instantiation

```
$\underline{(sk, PK) \leftarrow \texttt{FAAS-NTRU.Kg}(1^\kappa)}$:
1: Generate secret keys $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ such that $h(x) = p^{-1}g(x)f^{-1}(x)$
2: If $\mathbf{f}$ and $\mathbf{g}$ are not invertible mod $q$, go to Step 1
3: $sk' \leftarrow (\mathbf{f}, \mathbf{g})$, $PK' \leftarrow \mathbf{h}$ and $z \leftarrow \{0,1\}^\kappa$
4: Select integers $(k, t, b, L)$ as in generic $\texttt{FAAS.Kg}$ Step 1
5: $u_i \leftarrow H_1(i||z)$, $\boldsymbol{\beta_i} \leftarrow \texttt{pqNTRUsign.Sig}(u_i, sk')$, where $|u_i| = \kappa$ for $i = 0, \ldots, t-1$
6: Set pre-computed signature table $\mathbf{B} \leftarrow \{\boldsymbol{\beta}_i\}_{i=0}^{t-1}$
7: $\boldsymbol{\gamma}_i \leftarrow \texttt{pqNTRUsign.Sig}(i, sk')$, for $i = 0, \ldots, 2^b - 1$
8: Set pre-computed signature table $\boldsymbol{\Gamma} \leftarrow \{\boldsymbol{\gamma}_i\}_{i=0}^{2^b-1}$
9: $sk \leftarrow (z, \mathbf{B}, \boldsymbol{\Gamma})$ and $PK \leftarrow PK'$
```

```
$\underline{\sigma \leftarrow \texttt{FAAS-NTRU.Sig}(m, sk)}$:
1: $(j_1, \ldots, j_k) \leftarrow H_2(m||z)$, where each $\{j_i\}_{i=1}^k$ is interpreted as a $|t|$-bit integer.
2: $u_{j_i} \leftarrow H_1(j_i||z)$ and $(\mathbf{u_{p_{j_i}}}, \mathbf{v_{p_{j_i}}}) \leftarrow H_N(u_{j_i}||h)$ where $i = 1, \ldots, k$
3: $(\mathbf{u_p}, \mathbf{v_p}) \leftarrow \sum_{i=1}^k (\mathbf{u_{p_{j_i}}}, \mathbf{v_{p_{j_i}}})$, and $\boldsymbol{\sigma}_U \leftarrow \sum_{i=1}^k \beta_i$
4: $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||u)$, where each $\{j_i\}_{i=1}^L$ is interpreted as a $b$-bit integer.
5: $\mathbf{s} \leftarrow \boldsymbol{\sigma}_U + \sum_{i=1}^L \boldsymbol{\gamma}_{j_i^*}$ and $\boldsymbol{\sigma} \leftarrow (\mathbf{u_p}, \mathbf{v_p}, \mathbf{s})$
```

```
$\underline{\{0,1\} \leftarrow \texttt{FAAS-NTRU.Ver}(m, \sigma, PK)}$:
1: $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||u)$
2: $\hat{\mathbf{u}} = \boldsymbol{sh}^{-1} \mod \hat{q}$
3: if $||\hat{\mathbf{u}}||_\infty > \sqrt{(k+L)}\tau p \tilde{\sigma}$ then return 0
4: end if
5: $(\mathbf{u_{p_i}}, \mathbf{v_{p_i}}) \leftarrow H_N(j_i^*||h)$ where $i = 1, \ldots, L$
6: if $(\hat{\mathbf{u}}, \mathbf{s}) = (\mathbf{u_p}, \mathbf{v_p}) + \sum_{i=1}^L (\mathbf{u_{p_i}}, \mathbf{v_{p_i}})$ then return 1, else return 0
7: end if
```

for a prime q and a positive integer N [34]. We use a Bimodal Gaussian distribution $\chi_{\tilde{\sigma}}^N$ with standard deviation $\tilde{\sigma}$ to sample an N-dimension random vector $\mathbf{r}$. For `FAAS-NTRU`, we model a hash function $H_N : \{0,1\}^* \rightarrow \mathbb{Z}_q^N$. This enables generating random elements $\mathbf{m_p} = (\mathbf{u_p}, \mathbf{v_p})$ with $\mathbf{u_p} \in \mathbb{Z}_p^{N_1}$ and $\mathbf{v_p} \in \mathbb{Z}_p^{N_2}$ for a prime p and

$N = N_1+N_2$. We let $f(x), g(x), h(x) \in \mathcal{R}_q$ where $f(x)$ and $g(x)$ have small coefficients and $h(x) = p^{-1}g(x)f^{-1}(x)$. The NTRU lattice associated with $\mathbf{h}$ is defined as $\mathcal{L} = \{(\hat{\mathbf{u}}, \hat{\mathbf{v}}) \in \mathcal{R}_q^2 : \hat{\mathbf{u}}\mathbf{h} = \hat{\mathbf{v}}\}$. A vector in NTRU lattice can be written as $\mathbf{v} = \langle \hat{\mathbf{s}}, \hat{\mathbf{t}} \rangle$ where $\hat{\mathbf{s}}, \hat{\mathbf{t}} \in \mathcal{R}_q$, following [34], we refer to $\hat{s}$ as s-side and $\hat{t}$ as t-side of the vector. Hoffstein et al. also uses rejection sampling to ensure that the signature component does not leak any information about the private keys by checking if its norm is in $(-\frac{q}{2}+\nu_y, \frac{q}{2}-\nu_y)$ for some public parameter ν where $y \in \{s\text{-side}, t\text{-side}\}$. This is done as in the Step 4 of Algorithm 9. We also note that τ in Algorithm 9 is a Gaussian distribution parameter which ensures a bound on the value of the sampled vector's coordinates.

3.5 Security Analysis

Theorem 1 *$Adv^{A\text{-}EU\text{-}CMA}_{\texttt{FAAS},\mathcal{A}}(t, q_S, q_h)$ is bounded as follows.*

$$Adv^{A\text{-}EU\text{-}CMA}_{\texttt{FAAS},\mathcal{A}}(t, q_S, q_h) \leq Adv^{A\text{-}EU\text{-}CMA}_{\texttt{Asig},\mathcal{B}}(t', q'_S, q'_h)$$

where $t' = O(t) + 2q_S(t_{\texttt{RNG}} + t_{\texttt{Sig}} + t_{\texttt{Agg}})$, $q'_S \geq 2q_S$ and $q_H = q'_H$.

Proof 1 *Please refer to B*

3.6 Performance Evaluation and Comparison

We first give the analytical costs of `FAAS` instantiations and their counterparts in terms of computational overhead and key/signature sizes. We then outline our experimental setup, parameters and provide a detailed experimental comparison.

3.6.1 Analytical Performance

Computation Overhead: We analyze key generation, signature generation and verification overhead of `FAAS`, where the online costs can be seen in Table 3.1.

Key Generation (offline, once): `FAAS` instantiations require pre-calculating two tables in their key generation algorithm, and therefore, it is more expensive than their counterparts. Specifically, to generate the two pre-calculation tables, $2^b + t$ signatures of underlying aggregate signature schemes are computed.

Signature Generation: `FAAS` signature generation requires a small-constant number of signature aggregation, message aggregation and hash calls as $(k+L) \cdot ASig.\mathsf{Agg} + k \cdot H + k \cdot Agg$. Specifically, `FAAS-C-RSA` signature generation only requires a few modular multiplications over n and hash calls, while RSA needs a full exponentiation with a large integer d. pqNTRUsign requires Gaussian sampling and polynomial multiplication to generate a signature. This overhead is reduced to polynomial additions and mapping functions (H_N) in `FAAS-NTRU`. We present a variant of `FAAS-NTRU` (referred to as `FAAS-NTRU′`) to improve the efficiency of signature generation with the cost of an increased private key size. Our implementation (see Section 3.6.2) showed that the mapping function in `FAAS-NTRU` takes a significant amount of time. Therefore, to

Table 3.1: Analytical computation overhead analysis and comparison.

Schemes	**Signature Generation**	**Signature Verification**
RSA [49]	Exp_d	Exp_e
ECDSA [4]	$Emul + H + Mul_{q'}$	$1.3 \cdot Emul + Eadd + H$
BPV-ECDSA [18]	$v_B \cdot Eadd + H + Mul_{q'}$	$1.3 \cdot Emul + Eadd + H$
Ed25519 [14]	$Emul_{25519} + 2H + Mul_{q'}$	$1.3 \cdot Emul_{25519} + Eadd_{25519} + H$
SPHINCS [13]	$(2t_S - 1)H$	$(k_S((\log t_S) - x_S + 1) + 2^{x_S} - 1)H$
pqNTRUsign [34]	$Gsamp + Pmul$	$Pmul$
`FAAS`	$(k + L) \cdot ASig.\mathsf{Agg} + k \cdot H$ $+k \cdot Agg$	$ASig.\mathsf{Ver} + L \cdot Agg$
`FAAS-C-RSA`	$(2k + L) \cdot Mul_n + k \cdot H$	$Exp_e + L \cdot Mul_n$
`FAAS-NTRU`	$(2k + L) \cdot Padd + L \cdot H_N$	$Pmul + L \cdot Padd$
`FAAS-NTRU′`	$(2k + L) \cdot Padd$	$Pmul + L \cdot Padd$

$ASig.\mathsf{Agg}$ and $ASig.\mathsf{Ver}$ denote the cost of aggregate operation and aggregate signature verification of $ASig$, respectively. In our `FAAS` instantiations, the cost of message aggregation (with Agg) and signature aggregation with $ASig.\mathsf{Agg}$ are comparable (especially for `FAAS-C-RSA`). Exp_e and Exp_d denote an exponentiation with the small exponent e and large exponent d, respectively, over modulus n. $Emul$ and $Eadd$ denote the costs of EC scalar multiplication over modulus p', and EC addition over modulus p', respectively. $Emul_{25519}$ and $Eadd_{25519}$ are performed on twisted Edwards' curve. H and $Mul_{q'}$ denote a cryptographic hash and a modular multiplication over modulus q', respectively. We omit the constant number of negligible operations if there is an expensive operation (e.g., integer additions are omitted if there is an $Emul$ or Exp_e). We use double-point scalar multiplication for verifications of ECC based schemes ($1.3 \cdot Emul$ instead of $2 \cdot Emul$ [32]). $Pmul$ and $Padd$ denote polynomial multiplication and addition, respectively. $Gsamp$ represents the cost of Gaussian sampling over integers. H_N denotes the cost of the mapping hash function in pqNTRUSign. t_S, k_S and x_S are SPHINCS [13] parameters where t_S is the number of secret key elements, k_S is the number of revealed secret key elements and x_S is a small integer. k and L are `FAAS` parameters (as defined in Algorithm 7) and v_B is the BPV parameter and the recommended values are $k = L = v_B = 32$.

reduce the signature generation time, we offer a variant of `FAAS-NTRU` that stores the results of mappings as the private key (instead of deterministically generating them in the signature generation algorithm). Therefore, the cost of the mapping functions are avoided in `FAAS-NTRU′`.

Note that, aside from Gaussian sampling, pqNTRUsign also requires a rejection sampling in the signature generation to ensure that signatures do not leak sensitive information about the private key distribution. Therefore, due to rejection sampling, the signature generation of these schemes do not have a constant time, whereas

`FAAS` instantiations do not require rejection sampling.

Signature Verification and End-to-end Delay: `FAAS` instantiations add a slight overhead to the verification of their base schemes, which is equal to L message aggregations. However, note that, since message aggregation is highly efficient, this overhead is negligible in practice, especially considering the overall gain in terms of the end-to-end delay due to the highly efficient signature generation algorithm of `FAAS`.

ECDSA and Ed25519 require one scalar multiplication at the signer's and a *double scalar multiplication* at the verifier's side. Although, SPHINCS requires only hash calls, the number of hash calls at the signer is significantly high ($\approx 2^{17}$).

Discussion on Side-Channel Attacks: Side-channel attacks pose a serious threat to cryptographic implementations. Lattice-based cryptography offers efficient solutions with post-quantum security promise. However, most of the efficient lattice-based signature schemes require a (high precision) sampling from a distribution, mostly a Gaussian, which not only degrades their performance on the signer's side, but is also highly prone to side-channel attacks. For instance, BLISS [23], as one of the most efficient instances of such schemes, has been targeted with a number of side-channel attacks [30, 26]. Secure implementation approaches might mitigate some of these side-channel attacks; however, they are deemed to be a highly challenging and error prone task [24].

Table 3.1 shows that `FAAS` instantiations do not require any exponentiation or Gaussian sampling in signature generation algorithm. Although, these operations are required in the key generation of `FAAS` instantiations, they are done once and

Table 3.2: Analytical storage/transmission analysis and comparison.

Schemes	**Secret Key Size**	**Signature Size**	**Public Key Size**
RSA [49]	$\|n\| + \|d\|$	$\|n\|$	$\|n\| + \|e\|$
ECDSA [4]	$\|q'\|$	$\|q'\| + \|H\|$	$\|q'\|$
BPV-ECDSA [18]	$(n_B + 1) \cdot \|q'\|$	$\|q'\| + \|H\|$	$\|q'\|$
Ed25519 [14]	$\|q'\|$	$\|q'\| + \|H\|$	$\|q'\|$
SPHINCS [13]	n_S	$n_S(k_S(\|t_S\|$ $-x_S + 1) + 2^{x_S})$	n_S
pqNTRUsign [34]	$\|f\| + \|g\|$	$\|b\|$	$\|h\|$
FAAS	$(2^b + t + 1) \cdot \|ASig.\mathsf{Sig}\|$ $+\kappa$	$\|ASig.\mathsf{Sig}\| + \|Agg\|$	PK'
FAAS-C-RSA	$(2^b + t + 1) \cdot \|n\| + \kappa$	$2\|n\|$	$\|n\| + \|e\|$
FAAS-NTRU	$(2^b + t + 1) \cdot \|v'\| + \kappa$	$3\|v'\|$	$\|h\|$
FAAS-NTRU′	$(2^b + t + 1) \cdot \|v'\| + \kappa$ $t \cdot \|H_N\|$	$3\|v'\|$	$\|h\|$

b and n are FAAS parameters (as defined in Algorithm 7), and n_B is BPV parameter. Suggested values are $b = 8$ and $t = n_B = 256$. v' is the t-side of a vector where b is the s-side[34]. **h** is the public key of pqNTRUsign, where **f** and **g** are the secret keys of it [34]. SPHINCS [13] parameter n_S denotes the bit length of hashes. The sizes of each component are given in Table 3.4 for $\kappa = 128$.

offline. Therefore, FAAS instantiations offer an improved side-channel resiliency and easy implementation as compared to techniques that require online Gaussian sampling and exponentiation. Moreover, FAAS instantiations offer deterministic signing (as in Ed25519 [14]), that makes it immune to weak PRNG attacks (e.g. [35]).

Storage and Transmission Overhead: FAAS requires two pre-computation tables to be stored at the signer's side, with the size of $(2^b+t+1)\cdot|ASig.\mathsf{Sig}|+\kappa$. Moreover, in addition to their base scheme, FAAS requires an aggregated randomness which makes the total signature size $|ASig.\mathsf{Sig}|+|Agg|$. Note that, in FAAS-NTRU signature, the signature size changes from $|b|$ bits to $|v'|$ bits. The reason is that the 't-side' of the vector should be transmitted for aggregate verification in pqNTRUsign [34]. Therefore, the signature size of FAAS-NTRU increases slightly more. The public key size of FAAS instantiations is the same with that of their base aggregate signature schemes.

3.6.2 Performance Evaluation

Experimental Setup: We implemented FAAS instantiations on a laptop equipped with Intel i7 6700HQ 2.6GHz processor and 12 GB RAM. Our operating system was Linux Ubuntu 16.04 and we used gcc version 5.4.0.

Software Libraries and Implementation: We developed FAAS instantiations with C. We implemented FAAS-C-RSA with GMP library due to its optimized modular arithmetic operations [29]. We used the open-source pqNTRUsign implementation available in NTRU open source project [34] to develop FAAS-NTRU. We used Blake2 as our hash function (as in SPHINCS [13]), due to its high efficiency [8]. We open-source our implementations for wide-adaptation and comparison.

https://github.com/ozgurozmen/FAAS

We ran the open-source implementations of our state-of-the-art counterparts in our experimental setup, to present a fair comparison. We benchmarked the ECDSA in MIRACL library [52] and RSA in GMP library [29]. We implemented BPV pre-computation technique over MIRACL ECDSA implementation and benchmarked BPV-ECDSA. We benchmarked Ed25519 and SPHINCS using their Supercop implementations. Lastly, we used the open-source implementation of pqNTRUsign.

Parameters: We selected parameters to provide $\kappa = 128$-bit security. We selected $|n| = 3072$ bit, $|e| = 17$ bit and $|d| \approx 3072$ bit for RSA-based schemes. We chose $|p'| =$

$|q'| = 256$ bit for ECC-based schemes. We used the suggested parameters providing $\kappa = 128$-bit security for SPHINCS [13] and pqNTRUsign [34]. `FAAS` parameters are selected as $(b, L) = (8, 32)$ and $(k, t) = (256, 32)$ for $l_0 = l_2 = 256$. The security of these parameters depend on how many different combinations one can derive with k-out-of-t pre-computed components, that is $\binom{t}{k} = \binom{2^b}{L}$. With current parameters, there are 2^{141} different combinations that can be created. For instance, considering (k, t) parameters that are used for one-time masking, one can derive 2^141 different random values using these parameters. Another important aspect of security of (k, t) is to keep the indexes secret. As discussed in Section 3.4, this ensures that presented attack cannot be applied to `FAAS` schemes. Since we are concatenating a secret key in the hash call, the indexes will remain as secret. On the other hand, one can attack to H_0 and try to obtain an m^* such that $H_0(m||u)$ corresponds to the same indexes as $H_0(m^*||u)$. However, since u is a random value derived based on secret indexes, attacker must conduct a *target collision attack* to find such m^*. On the other hand, any permutation of the indexes would correspond to a collision on H_0. Since there are $k!$ different possible index permutations, the probability to find such an m^* is $\frac{L!}{2^{2^b}}$. With the current parameter selection, the probability for this is $\frac{1}{2^{138}}$. Since the underlying signature schemes' parameters are selected to provide $\kappa = 128$-bit security, all in all, `FAAS` instantiations offer $\kappa = 128$-bit security.

Experimental Comparison: Tables 3.3 and 3.4 give numerical evaluation and comparison of `FAAS` instantiations and their counterparts.

`FAAS` instantiations offer notably faster signature generation over their base schemes with a slightly slower verification. Specifically, `FAAS-C-RSA` signature generation

Table 3.3: Experimental performance comparison and analysis (in μ sec).

Schemes	**Signature Generation**	**Signature Verification**	**End-to-end Delay**	**Post-Quantum Promise**
RSA [49]	8083.26	47.74	8131.00	✗
ECDSA [4]	725.38	927.30	1652.68	✗
BPV-ECDSA [18]	149.60	927.30	1076.90	✗
Ed25519 [14]	132.61	335.95	468.56	✗
SPHINCS [13]	13458.18	370.50	13828.68	✓
pqNTRUsign [34]	14516.21	304.10	14820.31	✓
`FAAS-C-RSA`	**192.19**	61.97	**254.16**	✗
`FAAS-NTRU`	**489.29**	710.17	**1199.46**	✓
`FAAS-NTRU′`	**137.87**	710.17	**848.04**	✓

is over 40× faster than traditional RSA. `FAAS-NTRU` and `FAAS-NTRU′` also improve pqNTRUsign signature generation by 29.67× and 105.29×, respectively. However, `FAAS` instantiations require storing a secret key up to 1 MB (see Table 3.4). With their improved side-channel resiliency and fast signature generation, `FAAS` instantiations may be highly preferable for delay-aware applications where the signer can tolerate storing up to 1MB of private key. We observed that signature generation cost of `FAAS-NTRU` was dominated by the mapping functions, that are required to map the message to a vector. We further noticed that these vectors can be stored as the private key, instead of deterministically generating them at the signature generation. This resulted in a trade-off between signature generation time and secret key size, where signature generation speed up 3.55× with a 2× increased secret key (see Tables 3.3 and 3.4).

Recall that, SCRA [61] does not use a masking strategy, and therefore, leaks its private key (as shown in Section 3.3). Since `FAAS` uses a highly efficient and constant-

Table 3.4: Key/signature size comparison and analysis (in Bytes).

Schemes	**Secret Key Size**	**Signature Size**	**Public Key Size**
RSA [49]	768	372	386
ECDSA [4]	32	64	32
BPV-ECDSA [18]	10272	64	32
Ed25519 [14]	32	64	32
SPHINCS [13]	1088	41000	1056
pqNTRUsign [34]	1024	576	1024
`FAAS-C-RSA`	197408	768	386
`FAAS-NTRU`	525328	3072	1024
`FAAS-NTRU`$'$	1049600	3072	1024

size signature masking strategy, its signature generation requires only twice as much signature aggregations and k message aggregations compared to insecure SCRA. This results in an approximately three times increase in signature generation time. The signature verification of the both schemes are the same. Moreover, since `FAAS` relies on an efficient message encoding (see Section 3.4), the secret key of `FAAS` is $L\times$ smaller than that of SCRA. In practice, since L is selected as 32, this results in a significant improvement in terms of secret key size. Therefore, `FAAS` address the flaws of SCRA with a small computation overhead and a more compact secret key size.

3.6.3 Potential Impacts

Note that `FAAS` can be instantiated with any single signer aggregate signature scheme. In this thesis, we proposed some efficient instantiations, where signature verification was significantly faster than signature generation, which was optimized with `FAAS` strategy. For instance, like SCRA [61], `FAAS` can also be instantiated with

BGLS [15], that would offer low storage. However, since the verification requires pairing, an expensive operation, we do not present this instantiation. Moreover, we only instantiated `FAAS` with signature schemes that are proven to have secure aggregation. Signature aggregation has not been extensively studied in some signature schemes such as BLISS [23] and Dilithium [24]. BLISS can get benefit a lot from our `FAAS` construction due to its improved side-channel resiliency. Despite being one of the fastest signature schemes up to date, BLISS faced numerous side-channel attacks that pose a critical limit against its practical adoption. Dilithium and BLISS offer fast signature generation ($272\mu s$ and $141\mu s$, respectively, in our experimental setting) and even faster verification ($50\mu s$ and $28\mu s$, respectively, in our experimental setting). Furthermore, there are numerous schemes proposed to NIST Post-quantum competition, that requires Gaussian Sampling. Considering the side-channel resiliency of `FAAS` schemes by eliminating such operations in the online signing phase, we believe that `FAAS` can be widely used in practice to strengthen the aforementioned post-quantum schemes. However, first, the secure aggregation property of these schemes must be proven.

3.7 Related Work

In this section, we describe authentication mechanisms, schemes based on aggregate signatures and pre-computation methods that are most relevant to our work.

Message authentication codes (e.g. HMAC [36]) offer very fast authentication, however they lack scalability, non-repudiation and public verifiability. Although digital signatures offer these properties, they may be costly since they require expensive operations such as exponentiation [49] or scalar multiplication [4, 14]. One-time signatures (e.g. HORS [48], TV-HORS [58]) and delayed key disclosure methods (e.g. TESLA [46] have been proposed that offer fast signature generation and verification. However, HORS incurs large signature sizes, and a private/public key pair can only be used once. The continuous renewal and certification of the new keys might be impractical. TESLA requires time synchronization and packet buffering due to the delayed disclosure strategy. Thus, it cannot offer immediate verification, which might be essential for some use-cases. Online/offline signatures [51, 27, 44] offer very fast signing since they pre-compute tokens for each message to be signed at the offline phase. Therefore, at the online phase, these pre-computed tokens are used, that provides efficient signature generation. However, such methods incur linear storage with respect to the number of messages to be signed. Moreover, as tokens are depleted, they should be renewed continuously that might introduce further overhead. Therefore, we believe they may not be practical for real-time networks that require continuous, ultra-fast signature generation.

There are various schemes that leverage signature aggregation to ensure authentication and integrity in outsourced databases (e.g., [43, 41, 54]). In such applications, the signatures of a relatively small set of messages with well-defined indexes (e.g., signatures belonging to some row elements in a database table) are aggregated to obtain compact signatures for the response of basic database queries [43]. Despite their merits, potential security issues that may stem from the homomorphic properties of these signatures were pointed out [43, 38]. Specifically, it has been shown that, since aggregate signatures are mutable, one can create “new signatures" on data items that have not been explicitly queried by combining previously obtained aggregate signatures. To prevent this issue, immutable signatures (e.g., [43, 38]) have been developed, which generally rely on one-time masking and/or sentinel (umbrella) signatures. Recently, signature schemes that depend on secure aggregation (e.g. RA [60] and SCRA [61]) have been proposed. However, as discussed, RA [60] is an online/offline signature with a dependency on pre-defined structures in messages. In this thesis, we prove that SCRA leaks information about the private keys, that may cause serious consequences.

Traditional pre-computation methods (e.g., used in online/offline signatures [51, 27, 44]) can significantly decrease the computational overhead at the online phases. However, since they require linear storage overhead, their use in practice may be limited. Therefore, constant-storage pre-computation methods (as considered in this work) are proposed by Boyko-Peinado-Venkatesan [18]. This line of pre-computation also received a significant interest from research and a few variants of BPV that optimizes the storage/computation costs were later proposed in [7].

Chapter 4: Conclusion

Standard cryptographic suites offer high-security guarantees, but their high energy consumption poses an obstacle towards their broad adoption for battery-limited devices, which are an integral part of IoT applications (e.g., smart-home, healthcare).

In this thesis, we first develop a series of algorithmic improvements and optimizations that can be applied to a vast range of cryptographic techniques with only a minimal modification. It is central to our techniques to enable self-certification and small-constant size pre-computation capabilities for prominent key exchange, integrated encryption, and hybrid cryptographic constructions. We fully implemented our techniques and provided a comprehensive experimental evaluation of modern embedded systems to assess their practicality for real-life applications. Our experimental analysis confirmed up to 7$\times$ battery life improvements over the standard cryptographic techniques by introducing only a small-constant storage overhead. Our improvements adhere the core design properties of their base cryptographic standards, and can also be potentially adopted to other similar cryptographic techniques.

Inline with efficient digital signature schemes, we presented an attack to SCRA signature scheme that can recover the private key of the signer with an overwhelming probability by observing only 8192 signatures, for an instantiation. We then proposed a new generic signature scheme (i.e., `FAAS`) that can transform any single-signer (k-element extraction secure) aggregate signature into a signer efficient signature scheme with the cost of an increased private key size. Specifically, we proposed two instantiations of `FAAS` called `FAAS-C-RSA` and `FAAS-NTRU` that can offer up to 42$\times$ and 105$\times$ faster signature generation as compared to their base signature schemes, respectively. Moreover, our `FAAS-NTRU` instantiation provides a post-quantum promise with an improved side-channel resiliency.

Bibliography

[1] Ieee standard specifications for public-key cryptography - amendment 1: Additional techniques. *IEEE Std 1363a-2004 (Amendment to IEEE Std 1363-2000)*, pages 1–167, Sept 2004.

[2] Ieee standard communication delivery time performance requirements for electric power substation automation. *IEEE Std 1646-2004*, pages 1–24, 2005.

[3] Ieee guide for wireless access in vehicular environments (wave) - architecture. *IEEE Std 1609.0-2013*, pages 1–78, March 2014.

[4] American Bankers Association. *ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 1999.

[5] O. Arazi and H. Qi. Self-certified group key generation for ad hoc clusters in wireless sensor networks. In *Proceedings of IEEE International Conference on Computer Communications and Networks (ICCCN '05)*, pages 359–364, October 2005.

[6] G. Ateniese, G. Bianchi, Angelo Capossele, and Chiara Petrioli. Low-cost Standard Signatures in Wireless Sensor Networks: A Case for Reviving Pre-computation Techniques? In *Proceedings of NDSS 2013*, San Diego, CA, February 24-27 2013.

[7] Giuseppe Ateniese, Giuseppe Bianchi, Angelo T. Capossele, Chiara Petrioli, and Dora Spenza. Low-cost standard signatures for energy-harvesting wireless sensor networks. *ACM Trans. Embed. Comput. Syst.*, 16(3):64:1–64:23, 2017.

[8] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. Sha-3 proposal blake. Submission to NIST (Round 3), 2010.

[9] Mihir Bellare, JuanA. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 236–250. Springer Berlin Heidelberg, 1998.

[10] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[11] Daniel J. Bernstein. *Curve25519: New Diffie-Hellman Speed Records*, pages 207–228. Springer Berlin Heidelberg, 2006.

[12] Daniel J. Bernstein. Multi-user schnorr security, revisited. *IACR Cryptology ePrint Archive*, 2015:996, 2015.

[13] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical stateless hash-based signatures. In *Advances in Cryptology – EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 368–397. Springer Berlin Heidelberg, 2015.

[14] DanielJ. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[15] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proc. of the 22th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '03)*, pages 416–432. Springer-Verlag, 2003.

[16] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*, pages 416–432. Springer Berlin Heidelberg, 2003.

[17] Jurjen N. E. Bos and David Chaum. Provably unforgeable signatures. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 1–14. Springer Berlin Heidelberg, 1993.

[18] Victor Boyko, Marcus Peinado, and Ramarathnam Venkatesan. Speeding up discrete log and factoring based schemes via precomputations. In *Advances in Cryptology — EUROCRYPT'98: International Conference on the Theory and Application of Cryptographic Techniques*, pages 221–235. Springer Berlin Heidelberg, 1998.

[19] Certicom Research. Standards for efficient cryptography – SEC 1: Elliptic curve cryptography. `http://www.secg.org/download/aid-780/sec1-v2.pdf`, May 2009.

[20] J. Coron and D. Naccache. Boneh et al.'s k-element aggregate extraction assumption is equivalent to the diffie-hellman assumption. In *Proceedings of the 9th International Conference on the Theory and Application of Cryptology (ASIACRYPT 03')*, pages 392–397, 2003.

[21] Craig Costello and Patrick Longa. *FourQ: Four-Dimensional Decompositions on a Q-curve over the Mersenne Prime*, pages 214–235. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[22] Benedikt Driessen, Axel Poschmann, and Christof Paar. Comparison of innovative signature algorithms for wsns. In *Proceedings of the First ACM Conference on Wireless Network Security*, WiSec '08, pages 30–35. ACM, 2008.

[23] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013: 33rd Annual Cryptology Conference. Proceedings, Part I*, pages 40–56. Springer Berlin Heidelberg, 2013.

[24] Leo Ducas, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehle. Crystals – dilithium: Digital signatures from module lattices. Cryptology ePrint Archive, Report 2017/633, 2017. `https://eprint.iacr.org/2017/633`.

[25] Rachid El Bansarkhani and Johannes Buchmann. Towards lattice based aggregate signatures. In *Progress in Cryptology AFRICACRYPT 2014*, volume 8469 of *Lecture Notes in Computer Science*, pages 336–355. Springer International Publishing, 2014.

[26] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 1857–1874, 2017.

[27] S. Even, O. Goldreich, and S. Micali. Online/offline digital signatures. In *Proceedings on Advances in Cryptology (CRYPTO '89)*, pages 263–275. Springer-Verlag, 1989.

[28] S. Galbraith, J. Malone-Lee, and N.P. Smart. Public key signatures in the multi-user setting. *Information Processing Letters*, 83(5):263 – 266, 2002.

[29] Torbjörn Granlund. GNU multiple precision arithmetic library 6.1.2. `https://gmplib.org/`.

[30] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, gauss, and reload – a cache attack on the bliss lattice-based signature scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, pages 323–345. Springer Berlin Heidelberg, 2016.

[31] Isabelle Hang, Markus Ullmann, and Christian Wieschebrink. Short paper: A new identity-based dh key-agreement protocol for wireless sensor networks based on the arazi-qi scheme. In *Proceedings of the Fourth ACM Conference on Wireless Network Security*, WiSec '11, pages 139–144, New York, NY, USA, 2011. ACM.

[32] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[33] John Harding, Gregory Powell, Rebecca Yoon, Joshua Fikentscher, Charlene Doyle, Dana Sade, Mike Lukuc, Jim Simons, and Jing Wang. Vehicle-to-Vehicle Communications: Readiness of V2V Technology for Application. *U.S. Department of Transportation National Highway Traffic Safety Administration (NHTSA)*, August 2014.

[34] Jeffrey Hoffstein, Jill Pipher, William Whyte, and Zhenfei Zhang. A signature scheme from learning with truncation. Cryptology ePrint Archive, Report 2017/995, 2017. `https://eprint.iacr.org/2017/995`.

[35] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Cryptanalytic attacks on pseudorandom number generators. In Serge Vaudenay, editor, *Fast Software Encryption, FSE 1998*, pages 168–188. Springer Berlin Heidelberg, 1998.

[36] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. IETF RFC 2104, 1997.

[37] Hugo Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. Cryptology ePrint Archive, Report 2005/176, 2005. `http://eprint.iacr.org/2005/176`.

[38] D. Ma and G. Tsudik. A new approach to secure logging. *ACM Transaction on Storage (TOS)*, 5(1):1–21, 2009.

[39] Ken MacKay. micro-ecc: Ecdh and ecdsa for 8-bit, 32-bit, and 64-bit processors. Github Repository, 2013.

[40] S. S. Manvi, M. S. Kakkasageri, and D. G. Adiga. Message authentication in vehicular ad hoc networks: Ecdsa based approach. In *Proceedings of the 2009 International Conference on Future Computer and Communication*, ICFCC '09, pages 16–20, Washington, DC, USA, 2009. IEEE Computer Society.

[41] E. Mykletun, M. Narasimha, and G. Tsudik. Signature bouquets: Immutability for aggregated/condensed signatures. In *Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS '04)*, pages 160–176. Springer-Verlag, 2004.

[42] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. *Transaction on Storage (TOS)*, 2(2):107–138, 2006.

[43] E. Mykletun and G. Tsudik. Aggregation queries in the database-as-a-service model. In *Proceedings of the 20th IFIP WG 11.3 working conference on Data and Applications Security*, DBSEC'06, pages 89–103. Springer-Verlag, 2006.

[44] D. Naccache, D. M'Raïhi, S. Vaudenay, and D. Raphaeli. Can D.S.A. be improved? Complexity trade-offs with the digital signature standard. In *Proceedings of the 13th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '94)*, pages 77–85, 1994.

[45] Phong Q. Nguyen and Igor E. Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Designs, Codes and Cryptography*, 30(2):201–217, 2003.

[46] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient and secure source authentication for multicast. In *Proceedings of Network and Distributed System Security Symposium, NDSS 2001*, 2001.

[47] D. Pointcheval. Psec-3: Provably secure elliptic curve encryption scheme, 2000.

[48] L. Reyzin and N. Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In *Proceedings of the 7th Australian Conference on Information Security and Privacy (ACIPS '02)*, pages 144–153. Springer-Verlag, 2002.

[49] R.L. Rivest, A. Shamir, and L.A. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[50] Seung-Hyun Seo, Jongho Won, Elisa Bertino, Yousung Kang, and Dooho Choi. A security framework for a drone delivery service. In *Proceedings of the 2Nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, DroNet '16, pages 29–34. ACM, 2016.

[51] A. Shamir and Y. Tauman. Improved online/offline signature schemes. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 355–367, London, UK, 2001. Springer-Verlag.

[52] Shamus. Multiprecision integer and rational arithmetic c/c++ library (MIRACL). `https://github.com/miracl/MIRACL`. Last Accessed on 30/01/2018.

[53] A. Singla, A. Mudgerikar, I. Papapanagiotou, and A. A. Yavuz. Haa: Hardware-accelerated authentication for internet of things in mission critical vehicular networks. In *MILCOM 2015 - 2015 IEEE Military Communications Conference*, pages 1298–1304, Oct 2015.

[54] Wei Song, Bing Wang, Qian Wang, Zhiyong Peng, and Wenjing Lou. Tell me the truth: Practically public authentication for outsourced databases with multi-user modification. *Information Sciences*, 387:221 – 237, 2017.

[55] P. Szakacs-Simon, S. A. Moraru, and F. Neukart. Signal conditioning techniques for health monitoring devices. In *2012 35th International Conference on Telecommunications and Signal Processing (TSP)*, pages 610–614, July 2012.

[56] P. Szakacs-Simon, S. A. Moraru, and L. Perniu. Pulse oximeter based monitoring system for people at risk. In *2012 IEEE 13th International Symposium on Computational Intelligence and Informatics (CINTI)*, pages 415–419, Nov 2012.

[57] T. Tesfay and J. Y. Le Boudec. Experimental comparison of multicast authentication for wide area monitoring systems. *IEEE Transactions on Smart Grid*, PP(99), 2017.

[58] Q. Wang, H. Khurana, Y. Huang, and K. Nahrstedt. Time valid one-time signature for time-critical multicast data authentication. In *INFOCOM 2009, IEEE*, 2009.

[59] Jongho Won, Seung-Hyun Seo, and Elisa Bertino. A secure communication protocol for drones and smart objects. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 249–260. ACM, 2015.

[60] A. A. Yavuz. An efficient real-time broadcast authentication scheme for command and control messages. *IEEE Transactions on Information Forensics and Security*, 9(10):1733–1742, 2014.

[61] A. A. Yavuz, A. Mudgerikar, A. Singla, I. Papapanagiotou, and E. Bertino. Real-time digital signatures for time-critical networks. *IEEE Transactions on Information Forensics and Security*, 12(11):2627–2639, 2017.

[62] A. A. Yavuz and P. Ning. BAF: An efficient publicly verifiable secure audit logging scheme for distributed systems. In *Proceedings of 25th Annual Computer Security Applications Conference (ACSAC '09)*, pages 219–228, 2009.

[63] Chenxi Zhang, Pin-Han Ho, and Janos Tapolcai. On batch verification with group testing for vehicular communications. *Wireless Networking*, 17(8):1851–1865, November 2011.

[64] Y. Zheng. Digital signcryption or how to achieve cost(signature & encryption) $<<$ cost(signature) + cost(encryption). In *Proceedings of Advances in Cryptology (CRYPTO '97)*, pages 165–179, 1997.

[65] H. Zhu, F. Bao, and T. Chigan. Compact routing discovery protocol with lower communication complexity. In *IEEE Wireless Communications and Networking Conference (WNCN '06)*, 2006.

APPENDICES

Appendix A: Analytical Analysis

Analytical comparison of our techniques with their state-of-the-art counterparts are depicted in Table A.1. One may notice that the improvements enabled by our two-stage optimizations are: (i) BPV permitted us to reduce the cost of *Emul* operations to k *Eadd* (where $k = 8$ as in [6]), which offers significant performance gains. DBPV further amplified this gain by requiring slightly more storage (only possible with a small receiver set). (ii) The integration of certified ECDH via AQ enabled us to eliminate the transmission and verification of certificates for the initial key exchange operations.

We exemplified the impacts of these improvements over Fixed ECHMQV and ECIES schemes. Fixed ECHMQV required $7Emul$ performed by each node. Integrating AQ to ECHMQV, we eliminated the transmission of the certificate along with $2Emul$ computation required for its verification. With the help of BPV, another $2Emul$ were reduced to $2k$ *Eadd*. Hence, our improved fixed ECHMQV with AQ-BPV scheme only requires three full $3Emul$ along with *Eadd* operations. Similarly, ECIES takes the advantage of AQ by eliminating $2Emul$. We also integrated BPV and DBPV to ECIES, where each of them reduces the cost of one *Emul* to k *Eadd* on the sender side. Therefore, no full *Emul* is needed in the online phase of the sender. Moreover, the cost of Signcryption is also minimized at the sender side, where there is no *Emul* but only a few *Eadd*.

Table A.1: Analytical performance analysis of our schemes with their counterparts.

Protocol	Sender					Receiver	
	Private Key[†]	***Public Key***[†]	***Tag Size***	***Key exchange***[¶]	***Enc.+Sign / Enc.+MAC***	***Key exchange***[¶]	***Dec.+Sign / Dec.+MAC***
ECDSA+ECDH+Cert	$\|q\|$	$\|q\|$	$2\|q\|$	$4Emul + 2H + Eadd + 3Mulq$	-	$4Emul + 2H + Eadd + 3Mulq$	-
AQ	$2\|q\|$	$2\|q\|$	-	$2Emul + Eadd$	-	$2Emul + Eadd$	-
Fixed ECHMQV+Cert	$\|q\|$	$\|q\|$	$2\|q\|$	$7Emul + 5H + 2Eadd + 4Mulq$	-	$7Emul + 5H + 2Eadd + 4Mulq$	-
ECHMQV+Cert	$2\|q\|$	$2\|q\|$	$2\|q\|$	$3Emul + 3H + Eadd + Mulq$	-	$3Emul + 3H + Eadd + Mulq$	-
ECIES with ECDSA+Cert	$\|q\|$	$\|q\|$	$\|H\|$	-	$2Emul + 6H$	-	$Emul + 6H$
Signcryption with ECDSA+Cert	$\|q\|$	$\|q\|$	$\|q\| + \|H\|$	-	$Emul + 2H$	-	$2Emul + 2H + Eadd$
Our Proposed Improved Schemes with Optimization							
AQ-BPV	$\Gamma + 2\|q\|$	$\Gamma + 2\|q\|$	-	$\boldsymbol{Emul + (1+k)Eadd}$	-	$\boldsymbol{Emul + (1+k)Eadd}$	-
Fixed ECHMQV with AQ-BPV	$\Gamma + 2\|q\|$	$\Gamma + 2\|q\|$	-	$\boldsymbol{3Emul + 3H + (3+2k)Eadd + mulq}$	-	$\boldsymbol{3Emul + 3H + (3+2k)Eadd + mulq}$	-
ECHMQV with AQ-BPV	$\Gamma + 2\|q\|$	$\Gamma + 2\|q\|$	-	$\boldsymbol{2Emul + 3H + (2+k)Eadd + mulq}$	-	$\boldsymbol{2Emul + 3H + (2+k)Eadd + mulq}$	-
ECIES with AQ-BPV	$\Gamma + \|q\|$	$\Gamma + \|q\|$	$\|H\|$	-	$\boldsymbol{Emul + 6H + kEadd}$	-	$Emul + 6H$
ECIES with AQ-DBPV	$(r'+1)\Gamma + \|q\|$	$(r'+1)\Gamma + \|q\|$	$\|H\|$	-	$\boldsymbol{2kEadd + 6H}$	-	$Emul + 6H$
Signcryption with AQ-DBPV	$r'\Gamma + \|q\|$	$r'\Gamma + \|q\|$	$\|q\| + \|H\|$	-	$\boldsymbol{kEadd + 2H}$	-	$2Emul + 2H + Eadd$

¶ In designed variant of integrated protocols (i.e., ECIES, Signcryption), the sender knows receiver's public key and ID beforehand. $Emul$ and $Eadd$ denote the costs of EC scalar multiplication over modulus q and EC addition over modulus q, respectively. H and $Mulq$ denote a cryptographic hash and a modular multiplication over modulus q, respectively. k is the BPV parameter that shows how many pre-computed pairs are selected in the online phase. Suggested value for $k = 8$ [6]. r' is the constant number of public keys (servers) that the node will communicate.
† $\Gamma = n \cdot |q|$ where n is the number of pre-computed pairs. Parameter sizes for n, q and H are: $n = 160$, $|q| = 192$ bit, $|H| = 256$ bit

Appendix B: Proof of Theorem 1

Theorem 2 *$Adv^{A\text{-}EU\text{-}CMA}_{\texttt{FAAS},\mathcal{A}}(t, q_S, q_h)$ is bounded as follows.*

$$Adv^{A\text{-}EU\text{-}CMA}_{\texttt{FAAS},\mathcal{A}}(t, q_S, q_h) \leq Adv^{A\text{-}EU\text{-}CMA}_{\texttt{Asig},\mathcal{B}}(t', q'_S, q'_h)$$

where $t' = O(t) + 2q_S(t_{\texttt{RNG}} + t_{\texttt{Sig}} + t_{\texttt{Agg}})$, $q'_S \geq 2q_S$ and $q_H = q'_H$.

Proof 2 *We prove that if there exists an adversary $\mathcal{A}$ that can break the A-EU-CMA security of $\texttt{FAAS}$ signatures as in Definition 3 in time t, and after making q_S and q_H signature generation and hash queries, respectively, then one can use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ that can break the A-EU-CMA security of the underlying aggregate signature scheme ASig signatures as in Definition 3 in time t', and after making q'_S and q'_H signature generation and hash queries, respectively.*

Setup: $\mathcal{B}$ is initiated with the public key of the underlying aggregate signature scheme PK' where $(sk', PK') \leftarrow \texttt{Asig.Kg}(1^\kappa)$

- *$\mathcal{B}$ is provided with access to the $\texttt{A.Sig}_{sk}$ and $RO(\cdot)$ as defined in Section 3.2.*
- *$\mathcal{B}$ keeps tables (L_H, L_m) to keep track of random oracle and signature queries.*
- *$\mathcal{B}$ picks $z \xleftarrow{\$} \{0,1\}^\kappa$, sets $PK \leftarrow PK'$ and passes PK' to $\mathcal{A}$ as the public key of $\texttt{FAAS}$ signature scheme.*

Execute $\mathcal{A}^{\texttt{Asig}_{sk}(\cdot),RO(\cdot)}$: $\mathcal{B}$ replies $\mathcal{A}$'s signature and hash queries as follows.

- *To reply a signature queries on* m_i *for* $i \in (1, \ldots, q_S)$ $\mathcal{B}$ *works as follows:*

  1. $\mathcal{B}$ *computes* $(j_1, \ldots, j_k) \leftarrow$ *H-Sim*$(m_i||z, L_H, 2)$ *where each* $\{j_i\}_{i=1}^{k}$ *is interpreted as a* $|t|$*-bit integer, then computes* $(u_{j_1} \ldots u_{j_k}) \leftarrow$ *H-Sim*$(j_i||z, L_H, 1)$ *and* $u_i \leftarrow Agg(u_1, \ldots, u_k)$.
  2. $\mathcal{B}$ *queries* $\sigma_{u_i} \leftarrow$ $\texttt{SigA}_{sk'}(u_1, \ldots, u_k)$ *and stores* (u_i, σ_{U_i}) *in* L_m.
  3. $\mathcal{B}$ *computes* $(j_1^*, \ldots, j_k^*) \leftarrow$ *H-Sim*$(m_i||u_i, L_H, 0)$ *where each* $\{j_i^*\}_{i=1}^{L}$ *interpreted as a* $|b|$*-bit integer.*
  4. $\mathcal{B}$ *queries* $\sigma_{m_i} \leftarrow$ $\texttt{Asig.A}_{sk'}(j_1^*, \ldots, j_L^*)$ *and* $s \leftarrow$ $\texttt{Asig.Agg}(\sigma_{u_i}, \sigma_{m_i})$ *and stores* (m_i, σ_{m_i}) *in* L_m.
  5. $\mathcal{B}$ *returns* $\sigma_i = (u_i, s_i)$.

- *RO*$(\cdot)$ *Queries:* $\mathcal{B}$ *replies to* $\mathcal{A}$*'s queries on* H_0, H_1 *and* H_2 *hash functions on input* x *by initiating the H-Sim*(x, L_H, i) *oracle where* $i \in \{0, 1, 2\}$ *as defined in Section 3.2.*

Forgery of $\mathcal{A}$ *: After at most* q_S *signature queries,* $\mathcal{A}$ *outputs a forgery* $\langle m^*, (u^*, s^*)\rangle$ *under PK. As in Definition 3,* $\mathcal{A}$ *wins if* $(\texttt{FAAS.Ver}(m^*, \sigma^* = (u^*, s^*), PK) \wedge (m^* \cap L_m = \emptyset))$*. Otherwise,* $\mathcal{A}$ *loses in the experiment. If* $\mathcal{A}$ *loses in the experiment,* $\mathcal{B}$ *also loses and outputs 0.*

Forgery of $\mathcal{B}$ *: Given* $\mathcal{A}$*'s successful forgery* $\langle m^*, (u^*, s^*)\rangle$*,* $\mathcal{B}$ *works as follows.*

1. *First computes* $(\tilde{j}_1^*, \ldots, \tilde{j}_k^*) \leftarrow$ *H-Sim*$(m^*||z, L_H, 2)$ *and* $(u^*_{\tilde{j}_1^*}, \ldots, u^*_{\tilde{j}_k^*}) \leftarrow$ *H-Sim*$(\tilde{j}_i^*||z, L_H, 1)$ *and checks if* $u^* \overset{?}{=} Agg(u^*_{\tilde{j}_1^*}, \ldots, u^*_{\tilde{j}_k^*})$ *holds, it outputs 0 (this*

event can happen with a negligible probability since it is equivalent to breaking the hash function).

2. *$\mathcal{B}$ then computes $(j_1^*, \ldots, j_L^*) \leftarrow$ H-Sim$(m^* || u^*, L_H, 0)$ where each $\{j_i^*\}_{i=1}^{L}$ is interpreted as a $|b|$-bit integer.*

3. *$\mathcal{B}$ queries $\sigma_m \leftarrow \texttt{SigA}_{sk'}(j_1^*, \ldots, j_L^*)$, computes $\tilde{s}^* \leftarrow \texttt{ASig.Agg}(s^*, \texttt{Inv}(\sigma_m))$ (where $\texttt{Inv}$ is the inverting function based on the mathematical structure of the underlying aggregate signature scheme).*

4. *$\mathcal{B}$ outputs $(u^*, \tilde{s}^*)$ as a successful forgery of the underlying aggregate signature scheme ASig.*

Success Probability Analysis: We analyze the events that are needed for $\mathcal{B}$ to win the A-EU-CMA experiment as defined in Definition 3 for ASig as follows.

- *$\overline{Abort1}$: $\mathcal{B}$ does not fail in answering any of $\mathcal{A}$'s queries.*
- *$Forge$: $\mathcal{A}$ wins the A-EU-CMA experiment for $\texttt{FAAS}$.*
- *$\overline{Abort2}$: $\mathcal{B}$ does not abort during the forgery of $\mathcal{B}$.*
- *Win: $\mathcal{B}$ wins the A-EU-CMA experiment of $\texttt{Asig}$*

$\mathcal{B}$ wins if all the events happens and therefore, the probability $Adv_{\texttt{Asig},\mathcal{B}}^{\text{A-EU-CMA}}(t', q'_S, q'_h)$ decomposes as:

$$\Pr[Win] = \Pr[\overline{Abort1}] \cdot \Pr[Forge|\overline{Abort1}] \cdot \Pr[\overline{Abort2}|\overline{Abort1} \wedge Forge]$$

1. $\overline{Abort1}$: *$\mathcal{B}$ responds to each of $\mathcal{A}$'s sign queries by querying the* $\texttt{SigA}_{sk'}(\cdot)$ *as defined in Definition 3 twice. Therefore, $\mathcal{B}$ only aborts if it cannot receive a valid signature from the* $\texttt{SigA}_{sk'}(\cdot)$ *that only happens with a negligible probability, and therefore,* $\Pr[\overline{Abort1}] = 1$.

2. *Forge: $\mathcal{B}$ only aborts if adversary $\mathcal{A}$ aborts and since the simulation transcript is indistinguishable from that of the actual scheme (based on the discussion in the indistinguishability analysis), the probability that $\mathcal{B}$ does not abort and $\mathcal{A}$ wins the A-EU-CMA experiment is* $\Pr[Forge|\overline{Abort1}] = Adv^{\text{A-EU-CMA}}_{\texttt{FAAS},\mathcal{A}}(t, q_S, q_h)$.

3. $\overline{Abort2}$: *As highlighted in the Step 1 in Forgery of $\mathcal{B}$, the probability that $(u^*, \tilde{s}^*)$ is not a valid message-signature pair is negligible. Moreover, the probability that $u^* \cap L_m \rightarrow u^*$ happens is only $\frac{1}{2^\kappa}$, since it requires breaking the underlying hash function. Therefore, after the successful forgery by $\mathcal{A}$, $\mathcal{B}$'s forgery will also be valid and non-trivial with an overwhelming probability and it can be concluded that* $\Pr[\overline{Abort2}|\overline{Abort1} \wedge Forge] \approx 1$.

4. *Win: $\mathcal{B}$ wins the A-EU-CMA experiment of* $\texttt{Asig}$ *with probability denoted as* $\Pr[Win] = Adv^{\text{A-EU-CMA}}_{\texttt{Asig},\mathcal{A}}(t', q'_S, q'_h)$. *This event only happens when all the above events happen. This implies that the A-EU-CMA advantage of* $\texttt{FAAS}$ *adversary is bounded by the A-EU-CMA advantage of the underlying* $\texttt{Asig}$ *adversary.*

Execution Time Analysis: $\mathcal{B}$'s running time is that of $\mathcal{A}$'s plus all the the time it takes to respond $\mathcal{A}$'s queries. Each $\mathcal{A}$'s query requires two random number generator calls (which its cost is denoted by $t_{\texttt{RNG}}$) and two $\texttt{SigA}_{sk'}(\cdot)$ *calls. Each* $\texttt{SigA}_{sk'}(\cdot)$ *call corresponds to a* $\texttt{ASig.Sig}(\cdot)$ *and a* $\texttt{ASig.Agg}(\cdot)$ *calls, which cost $t_{\texttt{Sig}}$ and $t_{\texttt{Agg}}$,* *respectively. Therefore, $\mathcal{B}$'s running time is estimated as $t' = O(t) + 2q_S(t_{\texttt{RNG}} + t_{\texttt{Sig}} + t_{\texttt{Agg}})$.*

<u>Transcript Indistinguishability:</u> $\mathcal{A}$'s view of the actual scheme is the public key PK, the signatures $(\sigma_1, \ldots, \sigma_j)$ for $j \leq q_S$ and the output of the hash functions. $PK = PK'$ is an output of $\texttt{ASig.Kg}(1^\kappa)$, which is identical to the actual scheme. For the signatures $\{\sigma_j = (u_j, s_j)\}_{j=1}^{q_S}$, one can see that the distribution of u is identical to the actual scheme since it is the output of the $Agg(\cdot)$ algorithm on the values $\{u_i\}_{i=1}^{k}$ which have the same distribution as in the actual scheme, (due to the $RO(\cdot)$ calls). Moreover, the distribution of $\{s_i\}_{i=1}^{q_S}$ in the simulation is identical to those in the actual scheme, since they are all the output of the same $\texttt{ASig.Agg}(\cdot)$. Lastly, the output of the random oracles in the proof is simulated with the same domain for H_0, H_1 and H_2 as in the actual schemes.

Discussion: We note that for the instantiations in Section 3.4, the underlying scheme, C-RSA [41], is proven to be *k-element* secure as in Definition 4. As for the `FAAS-NTRU` instantiation [34], because of the probabilistic nature of signature generation due to the sampling step $r \leftarrow \chi_{\tilde{\sigma}}^N$, the aggregation of each $\mathbf{v} = \mathbf{v_1} + (-1)^b\mathbf{ag}$ where $\mathbf{u_1} = \mathbf{pr} + \mathbf{u_p}$, and $\mathbf{v_1} = \mathbf{u_1 h} \mod q$ leads to the aggregation of $\mathbf{r}$ in the aggregate signature. The aggregated randomness contributes to the hardness of the signature extraction problem since to do so, one needs to first take out the aggregated randomness from the signature.

We also note that since we are only aggregating 64 signatures, which is much less than even the theoretical bound mentioned in [34], `FAAS-NTRU` is immune to attack on batch signatures proposed in [34].