Practical Aggregate Signature from General Elliptic Curves, and Applications to Blockchain

Yunlei Zhao
School of Computer Science, Fudan University
Shanghai, China
ylzhao@fudan.edu.cn

ABSTRACT

Aggregate signature (AS) allows non-interactively condensing multiple individual signatures into a compact one. Besides faster verification, it is useful to reduce storage and bandwidth, and is especially attractive for blockchain and cryptocurrency. In this work, we first demonstrate the subtlety of achieving AS from general groups, by a concrete attack that actually works against the natural implementations of AS based on almost all the variants of DSA and Schnorr's. Then, we show that aggregate signature can be derived from the Γ-signature scheme proposed by Yao, et al. To the best of our knowledge, this is the first aggregate signature scheme from general elliptic curves without bilinear maps (in particular, the secp256k1 curve used by Bitcoin). The security of aggregate Γ-signature is proved based on a new assumption proposed and justified in this work, referred to as non-malleable discrete-logarithm (NMDL), which might be of independent interest. When applying the resultant aggregate Γ-signature to Bitcoin, the storage volume of signatures reduces about 49.8%, and the signature verification time can even reduce about 72%. Finally, we specify in detail the application of the proposed AS scheme to Bitcoin, with the goal of maximizing performance and compatibility. We adopt a Merkle-Patricia tree based implementation, and the resulting system is also more friendly to segregated witness and provides better protection against transaction malleability attacks.

CCS CONCEPTS

• **Security and privacy → Digital signatures**.

KEYWORDS

aggregate signature, blockchain, elliptic curves

ACM Reference Format:
Yunlei Zhao. 2019. Practical Aggregate Signature from General Elliptic Curves, and Applications to Blockchain. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS '19), July 9–12, 2019, Auckland, New Zealand.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3321705.3329826

AsiaCCS '19, July 9–12, 2019, Auckland, New Zealand
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6752-3/19/07…$15.00
https://doi.org/10.1145/3321705.3329826

1 INTRODUCTION

Bitcoin [51], with the introduction of the blockchain technology, was originally proposed by Nakamoto Satoshi in 2008. The key characteristics of blockchain consist in decentralization, openness, unforgeability, and anonymity. After about ten years of rapid development, blockchain has been more and more popular, and more applications are advocated into finance, healthcare, storage, education industries, etc. Nevertheless, there are still quite a lot of deficiencies to overcome. Taking Bitcoin as an example, below we review some deficiencies or bottlenecks it faces now.

Currently, due to the 1M-byte limitation of block size, about 7 transactions are conducted per second in the Bitcoin system. This leads to, in particular, longer confirmation latency, relatively higher transaction fees, and easier target of spam attacks [53].

As the crucial elements of a global consensus system, kept in check by the ability for every participant to validate all updates to the ledger, the size of signatures and the computational cost for verifying them are the primary limiting factors for its scalability [48]. Bitcoin uses the EC-DSA signature scheme [38] over the secp256k1 curve [23]. According to Bitcoin Stack Exchange, in a standard "*pay to public key hash*" (P2PKH) transaction or a "*pay to script hash*" (P2SH) transaction, the signatures occupy about 40% of transcript size.[1] In addition, an EC-DSA signature involves non-linear combination of ephemeral secret-key and static secret-key, which is the source for relative inefficiency and for the cumbersome in extensions to multi-signatures [9, 48], scriptless scripts [67], etc. As a consequence, recently there is also renewed interests in deploying Schnorr's signature with Bitcoin in the future.

Aggregate signature (AS) [18] can essentially mitigate the above deficiencies or bottlenecks faced by Bitcoin (and actually almost all blockchain-based systems). An AS scheme is a digital signature scheme with the following additional property: multiple individual signatures $\{\sigma_1, \cdots, \sigma_n\}$, where σ_i is a signature on message m_i under public-key pk_i, $1 \leq i \leq n$ and $n \geq 2$, can be *non-interactively* collected and condensed into a compact aggregate signature σ. Here, in general, for any i, j such that $1 \leq i \neq j \leq n$, it is assumed that $(pk_i, m_i) \neq (pk_j, m_j)$. There is a corresponding aggregate verification process that takes input $\{(pk_1, m_1), \cdots, (pk_n, m_n), \sigma\}$, and accepts if and only if all the individual signatures are valid. Aggregate signature is useful to reduce bandwidth and storage volume, and is especially attractive for blockchain where communication and storage are more expensive than computation.

[1] In more detail, for a standard P2PKH or P2SH transaction with n inputs and m outputs, its size is about $146n + 33m + 10$ bytes where the signatures occupy $72n$ bytes. For P2SH multi-signature transactions, the size of signatures may further scale up.

The differences between aggregate signature and multi-signature should be noted. With a multi-signature scheme [4, 9, 14, 17, 35, 36, 40, 41, 43, 48, 49, 52, 58], multiple signers sign the same message, and more importantly they need interactive cooperation. Practical multi-signature schemes were built from general groups on which the discrete logarithm problem is hard [9, 48], in the plain public-key model where no trusted setup or proof-of-possession of secret key is needed. However, the known efficient aggregate signature schemes in the plain public-key model were all built from gap groups with *bilinear maps* [8, 18]. Aggregate signatures can also be built assuming: signer cooperation and interaction [41, 42], or trusted setup [49], or proof-of-possession of secret key [58], or synchronous communications [35]. But these assumptions are, in general, less realistic for decentralized blockchain systems like the Bitcoin.

1.1 Contributions

In this work, we investigate the applicability of the Γ-signature scheme proposed by Yao and Zhao [70]. Akin to Schnorr's, Γ-signature is generated with linear combination of ephemeral secret-key and static secret-key, and enjoys almost all the advantages of Schnorr's signature. Besides, Γ-signature has advantageous features in online/offline performance, stronger provable security, and deployment flexibility with interactive protocols like IKE. In this work, we identify one more key advantage of Γ-signature in signature aggregation, which is particularly crucial for applications to blockchain and cryptocurrency.

We first demonstrate the subtlety of achieving aggregate signatures from general elliptic curves (EC). This is illustrated with a concrete attack against a natural implementation of aggregating Schnorr's signatures. The attack is a type of *ephemeral* rogue-key attack, and actually works against the natural implementations of AS from almost all the variants of DSA and Schnorr's. It serves as a good warm-up for achieving aggregate signature from general EC groups without bilinear maps.

Then, we show that aggregate signature can be derived from the Γ-signature scheme. To the best of our knowledge, this is the first aggregate signature scheme from general groups without bilinear maps in the plain public-key model. The security of aggregate Γ-signature is proved based on a new assumption proposed and justified in this work, referred to as non-malleable discrete-logarithm (NMDL), which might be of independent interest. We provide the implementation of aggregate Γ-signature, with source code (anonymously) available from github [2]. When applying the resultant aggregate Γ-signature to Bitcoin, the storage volume of signatures reduces about 49.8%, and the signature verification time can even reduce about 72%.

Finally, we specify in detail the implementation of aggregate Γ-signature for Bitcoin. The goal is to maximize performance and compatibility with the existing Bitcoin system. Towards this goal, we adopt a Merkle-Patricia tree (MPT) aided implementation of our aggregate signature scheme. The implementation only brings minimal modifications, which are, in turn, more friendly to segregated witness (SegWit), and provide better protection against transaction malleability attacks [20].

2 PRELIMINARIES

For prime number q, denote by Z_q the additive group of integers modulo q, by Z_q^* the multiplicative group of integers modulo q. If $\mathcal{S}$ is a finite set then $|\mathcal{S}|$ is its cardinality, and $x \leftarrow S$ is the operation of picking an element uniformly at random from S. If α is neither an algorithm nor a set then $x \leftarrow \alpha$ is a simple assignment statement. A string or value α means a binary one, and $|\alpha|$ is its binary length. If α and β are two strings, $\alpha||\beta$ is their concatenation. If $\mathcal{A}$ is a probabilistic algorithm, $\mathcal{A}(x_1, x_2, \cdots ; \rho)$ is the result of running $\mathcal{A}$ on inputs $x_1, x_2, \cdots$ and random coins (i.e., random bits) ρ. Let $y \leftarrow \mathcal{A}(x_1, x_2, \cdots ; \rho)$ denote the experiment of picking ρ at random and letting y be $A(x_1, x_2, \cdots ; \rho)$. By $\Pr[E : R_1; \cdots ; R_n]$ we denote the probability of event E, after the *ordered* execution of random processes $R_1, \cdots , R_n$. A function $\varepsilon(l)$ is *negligible* if for every $c > 0$ there exists an l_c such that $\varepsilon(l) < \frac{1}{l^c}$ for all $l > l_c$. Let *PPT* stand for probabilistic polynomial-time.

A digital signature scheme consists of three algorithms *KeyGen*, *Sign* and *Verify*, where the key generation algorithm *KeyGen* takes a security parameter l as input and randomly outputs a key pair (sk, pk). The signature algorithm *Sign* takes sk, m as input and outputs a signature σ. The signature verification algorithm *Verify* takes pk, m, σ as input and outputs *ACCEPT* or *REJECT*. Usually, the algorithms *KeyGen* and *Sign* are probabilistic, while the algorithm *Verify* is deterministic. The completeness of a signature scheme requires that $Verify(pk, m, Sign(sk, m)) = ACCEPT$ always holds for any $m \in \{0, 1\}^*$, as long as (sk, pk) is a valid key pair generated by running *KeyGen*.

2.1 Elliptic Curve for Bitcoin

We consider signature implementations over elliptic curve groups. Let $E(F)$ be the underlying elliptic curve group defined over finite field F, and the point P generates a cyclic group of prime order q on which the discrete logarithm problem is assumed to be hard, where $|q| = l$ is the security parameter. The order of $E(F)$ is tq, where t is called the cofactor that is usually a small constant. Denote by ∞ the identity element in $E(F)$.

Bitcoin uses the secp256k1 curve [23]: $y^2 = x^3 + 7$ defined over F_p for prime number $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$. For the secp256k1 curve, both p and q have the same length of 256 bits, i.e., $l = \log q = 256$, and the cofactor $t = 1$. For a point on the secp256k1 curve, it can be represented with 257 bits as (x, b), where $x \in Z_p$ is its x-coordinate and $b \in \{0, 1\}$ indicates the sign of its y-coordinate. Thanks to the fact that $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 = 7 \mod 8$, recovering y from (x, b) is very efficient for the secp256k1 curve [34, 44]. We remark that compact representation of public key has already been being employed in the Bitcoin system.

2.2 Schnorr Signature

The Schnorr signature scheme is proposed in [61], and is proven secure in the random oracle model based on the discrete logarithm assumption [55]. At a high level, Schnorr's signature is an instantiation of the Fiat-Shamir transformation [28] being applied to Σ-protocols (i.e., three-round public-coin honest-verifier zero-knowledge protocols) in the random oracle model. Let $H : \{0, 1\}^* \longrightarrow$

[2] https://github.com/AggregateGammaSignature/source

Z_q be a cryptographic hash function, and $m \in \{0,1\}^*$ be the message to be signed, Schnorr's signature scheme is briefly reviewed in Table 1.

KeyGen(1^l)	**Sign**(X, x, m)	**Verify**$(X, m, \sigma = (e, z))$
$x \leftarrow Z_q^*$	$r \leftarrow Z_q$	$R := zP - eX$
$X := xP$	$R := rP$	**if** $H(R, m) \neq e$ **then**
return(x, X)	$e := H(R, m)$	**return** *REJECT*
	$z := r + ex \mod q$	**else**
	return $\sigma = (e, z)$	**return** *ACCEPT*

Table 1: Schnorr's signature

2.3 Γ-Signature

Under the motivation for achieving signature schemes of better online/offline performance, flexible and easy deployments (particularly with interactive protocols like IKE), and stronger security, Yao and Zhao introduced a new paradigm in [70]. Specifically, they proposed a special case of Σ-protocol, which is referred to as Γ-protocol, and a transformation called Γ-transformation that transforms any Γ-protocol into a signature scheme in the random oracle model. The resultant signature is named Γ-signature. Below, we briefly review the Γ-signature scheme based on discrete logarithm problem (DLP), and its security result. The reader is referred to [70] for more details.

Let $H_d, H_e : \{0,1\}^* \rightarrow Z_q^*$ be two cryptographic hash functions, and $m \in \{0,1\}^*$ be the message to be signed, the DLP-based Γ-signature scheme is briefly reviewed in Table 2 (page 6). Here, for presentation simplicity, checking $z \neq 0$ in signature generation, and checking $d, z \in Z_q^*$ and $A \neq \infty$ in signature verification, are not explicitly specified. In the actual implementation, it is also suggested in [70] that $d = H_d(A)$ is replaced with $d = x_A \mod q$, where x_A is the x-coordinate of A. To ease signature verification, we can replace d in σ with d^{-1}. In this case, d^{-1} is not needed to computed in signature verification, and the signature is rejected if $H_d(A)d^{-1} \neq \infty$.

Security of Γ-Signature. Strong existential unforgeability under concurrent interactive attacks for a signature scheme $\prod = (KeyGen, Sign, Verify)$, *where a signature can be divided into two parts* (d, z), is defined using the following game between a challenger and a forger adversary $\mathcal{F}$.

Setup On the security parameter l, the challenger runs $(PK, SK) \leftarrow KeyGen(1^l)$. The public-key PK is given to adversary $\mathcal{F}$ (while the secret-key SK is kept private).

Concurrent interactive oracle access On the security parameter l, suppose $\mathcal{F}$ makes at most q_s signature queries to the signing oracle $Sign(SK, \cdot)$, where q_s is polynomial in l. Suppose $\mathcal{F}$ makes at most q_s signature queries. Each signature query consists of the following steps: (1) $\mathcal{F}$ sends "Initialize" to the signer. The i-th initialization query is denoted as I_i, $1 \leq i \leq q_s$. (2) Upon the i-th initialization query, the signer responds back d_i. (3) $\mathcal{F}$ adaptively chooses the message m_i to be signed, and sends m_i to the signer. (4) The signer sends back z_i, where (d_i, z_i) is the signature on message m_i. $\mathcal{F}$ is allowed to adaptively and concurrently interact with the signer in arbitrary interleaved order. As a special case, $\mathcal{F}$ can first make q_s initialization queries, and get all the values in $\bar{D} = \{d_1, \cdots, d_{q_s}\}$ before presenting any message to be signed.

Output Finally, $\mathcal{F}$ outputs a pair of m and (d, z), and wins the game if (1) $Verify(PK, m, d, z) = 1$ and (2) $(m, d, z) \notin \{(m_1, d_1, z_1), \cdots, (m_{q_s}, d_{q_s}, z_{q_s})\}$.

We define $\mathsf{AdvSig}^{\text{suf-cia}}_{\prod, \mathcal{F}}(1^l)$ to be the probability that $\mathcal{F}$ wins in the above game, taken over the coin tosses of $KeyGen$, $\mathcal{F}$, and the signer (and the random choice of the random oracle). We say the signature scheme $\prod$ is strongly existential unforgeable, if $\mathsf{AdvSig}^{\text{suf-cia}}_{\prod, \mathcal{F}}(\cdot)$ is a negligible function for every PPT forger $\mathcal{F}$.

It is proved in [70] that the above Γ-signature scheme is *strongly existential unforgeable* under the DLP assumption, assuming H_d is a random oracle while H_e is target one-way as defined in [70]. Roughly speaking, H_e is target one-way w.r.t. an e-condition R_e, if for any PPT algorithm $A = (A_1, A_2)$ it holds that $\mathsf{Adv}^{\text{tow}}_{H_e, A}(1^l) = \Pr[R_e(d, e = H_e(m), d', e' = H_e(m')) = 0 : d \leftarrow Z_q^*; (m, s) \leftarrow A_1(H_e, d); d' \leftarrow Z_q^*; m' = A_2(H_e, d, m, d', s)]$ is negligible, where s is some state information passed from A_1 to A_2. Here, the e-condition is defined as $R_e(d, e, d', e') = 0$ iff $d^{-1}e = d'^{-1}e' \mod q$. Introducing target one-wayness in [70] is to mitigate the dependency of provable security on random oracles. Specifically, for the two hash functions H_d and H_e used for Γ-signature, only H_d is assumed to be a random oracle. Detailed discussions on target one-way hash, including clarifications on the relations among target one-wayness, collision resistance and preimage resistance, are presented in [70], which show target one-wayness is a natural and realistic property for cryptographic hash functions. In particular, target one-wayness is implied by random oracle [70].

As a part in the security proof of Γ-signature [71], we have the following corollary:

Corollary 2.1. *The signing oracle* $Sign(SK, \cdot)$ *in the stage of "concurrent interactive oracle access" can be statistically simulated by a PPT algorithm* $\mathcal{S}$ *in the random oracle model. Specifically, the view of* $\mathcal{A}$ *when concurrently interacting with the signing oracle* $Sign(SK, \cdot)$ *is statistically close to that under the simulation of* $\mathcal{S}$.

3 AGGREGATE SIGNATURE AND MOTIVATION

An aggregate signature (AS) scheme is a tuple $(KeyGen, Sign, Verify, Agg, AggVerify)$, where the last three are deterministic, while the first three algorithms constitute a standard signature scheme. Given multiple individual signatures $\{\sigma_1, \cdots, \sigma_n\}$, where σ_i is a signature on message m_i under public-key pk_i, $1 \leq i \leq n$ and $n \geq 2$, the aggregation algorithm Agg condenses them into a compact aggregate signature sig. Here, in general, for any i, j such that $1 \leq i \neq j \leq n$, it is assumed that $(pk_i, m_i) \neq (pk_j, m_j)$; but it might be the case that $pk_i = pk_j$ or $m_i = m_j$. The completeness of an AS scheme says that $AggVerify(\{(pk_1, m_1), \cdots, (pk_n, m_n)\}, sig)$ returns "ACCEPT", whenever $Verify(pk_i, m_i, \sigma_i)$ outputs "ACCEPT" for any i, $1 \leq i \leq n$. Roughly speaking, the security of an AS scheme says that it is infeasible for any PPT adversary $\mathcal{A}$ to produce a valid forged aggregate signature involving an honest signer, even when it can play the role of all the other signers (in particular choosing their public keys), and can mount a chosen-message attack on the target honest signer.

Definition 3.1 (security of aggregate signature). Let $(pk, sk) \leftarrow KeyGen(1^l)$ be the public and secret key pair of the target honest signer. The advantage of the attacker $\mathcal{A}$ against the AS scheme is defined as $Adv_{AS}^{\mathcal{A}}(1^l) = Pr[AggVerify(\{(pk_1, m_1), \cdots, (pk_n, m_n)\}, sig) = ACCEPT]$, where n is polynomial in l and $pk \in \{pk_1, \cdots, pk_n\}$. The probability is taken over the random coins used by $KeyGen$ and $\mathcal{A}$ in the following experiment:

$$(pk, sk) \leftarrow KeyGen(1^l); (pk_1, ..., pk_n, m_1, ..., m_n, sig) \leftarrow \mathcal{A}^{Sign(sk,)}(1^l, pk) .$$

To make the security definition meaningful, we only consider adversaries that are legitimate in the sense that, supposing $pk = pk_i$ for some i, $1 \leq i \leq n$, $\mathcal{A}$ must never have queried m_i to its signing oracle. Then, an AS scheme is said to be secure if for any PPT adversary $\mathcal{A}$ its advantage $Adv_{AS}^{\mathcal{A}}(1^l)$ is negligible in l. Note that $\mathcal{A}$ can choose $pk_1, ..., pk_n$ as it wishes, in particular as a function of the target public key pk. There is also no requirement that the adversary "knows" the secret key corresponding to a public key it produces.

There were some discussions on deploying the pairing-based AS scheme proposed in [18] in the Bitcoin system [47], which are briefly summarized below.

- System complexity. Deploying pairing-based aggregate signature schemes requires the replacement of not only the EC-DSA algorithm but also the underlying elliptic curve. It makes a deployment in practice (such as Bitcoin) much more invasive than simply shifting algorithms.
- Bilinear group vs. general group. Intractability problems in groups with bilinear maps are weaker than the discrete logarithm problem in general EC groups.
- Verification speed. As an individual signature scheme, the verification of the pairing-based BLS signature is significantly slower than that of EC-DSA. Note that the miners still need to verify the correctness of individual BLS signatures before aggregating them into a block. Some survey indicates that on a concrete hardware it can verify 70,000 secp256k1 signatures per second, while it could only verify about 8,000 BLS signatures per second [47]. And the situation may be much further worsen, because according to the latest work [6], the size of element in G_1 is suggested to be 460-bit to satisfy 128-bit security, for the pairing $G_1 * G_2 \longrightarrow G_T$ based on the popular BN curve.

It is thus highly desirable to develop aggregate signatures, with the following features simultaneously:

- It can be built from general elliptic curves (without bilinear maps), in the plain public-key model with fully asynchronous communications.
- The underlying signature scheme has provable security, and moreover, is more efficient and flexible than EC-DSA.

4 SUBTLETY AND WARM-UP FOR ACHIEVING AS FROM GENERAL ELLIPTIC CURVES

Recently, there is renewed interests in deploying Schnorr's signature in the Bitcoin system, for its efficiency and flexibility. In comparison with EC-DSA used in Bitcoin, the linear combination of ephemeral secret-key and static secret-key with Schnorr's signature brings more desirable advantages, e.g., multi-signature, scriptless scripts (specifically, privacy-preserving smart contracts). However, we show the subtlety of aggregating Schnorr's signatures. This is demonstrated by a concrete fatal attack, which actually works against the natural implementations of aggregate signature based upon almost all the variants of DSA and Schnorr's.

We first present the aggregate signature based on Schnorr's scheme. Suppose there are n signers, $n \geq 2$, and each has the public and secret key pair (X_i, x_i) where $X_i = x_i P$ and $x_i \leftarrow Z_q^*$, $1 \leq i \leq n$. Denote by $\sigma_i = (e_i, z_i)$ the signature by user i on message $m_i \in \{0, 1\}^*$. After receiving $\{(X_1, m_1, \sigma_1), \cdots, (X_n, m_n, \sigma_n)\}$, the miner first verifies the correctness of each individual signature (X_i, m_i, σ_i), during which it gets $R_i = z_i P - e_i X_i$. If all the individual signatures are correct, the miner finally outputs $\hat{R} = \{R_1, \cdots, R_n\}$ and $z = \sum_{i=1}^n z_i$ as the resultant aggregate signature. On input $(X_1, \cdots, X_n, m_1, \cdots, m_n, \hat{R}, z)$, $AggVerify$ works as follows: computes $e_i = H(X_i, R_i, m_i)$, and accepts if $zP = \sum_{i=1}^n R_i + \sum_{i=1}^n e_i X_i$.

The above aggregate signature scheme looks fine. But a careful examination divulges the following subtle yet fatal attack. Without loss of generality, suppose the index of the attacker is 1, who possesses the public and secret key pair (X_1, x_1) and acts as follows.

- For any j, $2 \leq j \leq n$, the attacker selects m_j and arbitrary R_j (from the underlying EC group) on behalf of X_j, and computes $e_j = H(X_j, R_j, m_j)$. Note that the attacker does not necessarily know the discrete logarithm of either X_j or R_j for $2 \leq j \leq n$.
- The attacker chooses its own message m_1, sets the ephemeral rogue-key $R_1 = (-\sum_{j=2}^n R_j - \sum_{j=2}^n e_j X_j)$, and computes $e_1 = H(X_1, R_1, m_1)$ and $z = e_1 x_1$.
- Finally, it outputs $(R_1, \cdots, R_n, z)$ as the forged aggregate signature.

Note that $zP = e_1 x_1 P = e_1 X_1 = e_1 X_1 + R_1 + (\sum_{j=2}^n R_i + \sum_{j=2}^n e_i X_i) = \sum_{i=1}^n R_i + \sum_{i=1}^n e_i X_i$. Thus, the forged aggregate signature is valid, the attacker can sign arbitrary messages on behalf of the victim users $(X_2, \cdots, X_n)$. There is no doubt that such an attack is really fatal, particularly for a cryptocurrency system like Bitcoin. To stop such an attack, one approach is to require proof of possession of the discrete logarithms for all the R_j's, $1 \leq j \leq n$ [58]. But this voids the advantage of signature aggregation.

We suggest that the above ephemeral rogue-key attack might implicitly account for the reason why no previous AS scheme was built from general elliptic curve groups, *though we are unaware of any explicit presentation of such an attack to the best of our knowledge.* Nevertheless, it indeed serves as a good illustration of the subtlety of, as well as warm-up for, achieving AS from general groups without bilinear maps.

5 AGGREGATE Γ-SIGNATURE

The aggregate Γ-signature scheme is described in Table 2, where in the specification of *Agg* algorithm A_i is generated when computing $Verify(X_i, m_i, \sigma_i)$. Here, the algorithms $(KeyGen, Sign, Verify)$ just constitute the Γ-signature scheme presented in Section 2.3. For presentation simplicity, we use a single cryptographic hash function $H : \{0,1\}^* \rightarrow Z_q^*$, and the checking of $d, z \in Z_q^*$, $m_i \neq \lambda$ and $A \neq \infty$ is omitted in the specification of verification algorithms, where λ represents the empty string. The completeness property can checked directly.

Given a list of individual signatures $\{(X_1, m_1, \sigma_1 = (d_1, z_1)), \cdots, (X_n, m_n, \sigma_n = (d_n, z_n))\}$, where $n \geq 2$, the aggregation algorithm discards (X_i, m_i, σ_i) if the signature verification fails, or any one of (X_i, m_i) or A_i is repeated, where A_i is generated when computing $Verify(X_i, m_i, \sigma_i)$, $1 \leq i \leq n$. The latter checking is for provable security, as we shall see. But it still might be the case that, for some $i \neq j$, $X_i = X_j$ or $m_i = m_j$ (this case occurs with Bitcoin P2SH multi-signature transactions). We assume that the elements in $\hat{T}$ and those in $\hat{A}$ output by *Agg* are sorted to ease verification of aggregate signature. More details about the implementations are discussed in the next subsection. Observe that $\hat{T}$ and $\hat{A}$ are output and treated *separately*, and $AggVerifier$ actually does not care about the correspondence between the elements in $\hat{A}$ and those in $\hat{T}$. This flexibility allows for implementations more friendly to SegWit and to being resistant to transaction malleability attacks, as we shall discuss in Section 6.

The total size of the aggregate signature $(\hat{A}, z)$ has $n'(l+1) + l$ bits, where each A_i is represented with $\log p + 1 = l + 1$ bits. In comparison, the total size of n' individual signatures has $2n'l$ bits. For Bitcoin, $l = 256$, and n' is about 4000 on average. Thus, with aggregate Γ-signature, the storage volume of signatures reduces about 49.8%.

We use the simultaneous point multiplication techniques [27, 31, 34] in computing $zP + \sum_{j=1}^{n'} e_j X_j - \sum_{j=1}^{n'} d_j A_j$. Specifically, we divide the $2n'+1$ point multiplications into $\lceil (2n'+1)/8 \rceil$ groups, and then apply the simultaneous multiplication technique to each group of at most 8 point multiplications. Denote by A (resp., D) the timing cost for performing modular EC addition (resp., doubling), where 1D amounts to about 0.7A (with some optimization techniques, addition can be as efficient as doubling, i.e., 1A can amount to 1D). The cost for performing 8 point multiplications *separately* is about $8l$D + $4l$A. In comparison, for performing 8 point multiplication *simultaneously*, the cost is about l(D + A) plus at most 256A (for preparing a table of size at most $2^8 = 256$). This way, the timing cost for verifying signatures can reduce about 72%.

5.1 NMDL Assumption, and Justification

Motivated for breaking some impossibility barriers of black-box cryptography and for achieving cryptographic schemes of conceptually simple structure and analysis, the research community has been paying more attention to achieving cryptographic schemes based on non-black-box assumptions or primitives in recent years [5, 16, 21, 22, 25, 30]. As a popular non-black-box assumption, the knowledge-of-exponent assumption (KEA) and its variants have been shown to be successful and powerful (see, e.g., [1, 10, 11, 16, 21, 22, 24–26, 29, 30, 32, 33, 39, 50, 56, 57, 59, 60, 68, 69, 71]). In particular, a type of KEA assumption on pairing groups is used in Zcash [59, 60].

Yao and Zhao introduced and justified a variant of the KEA assumption, referred to as *joint* KEA (JKEA) assumption [69]. Let $H_1, \cdots, H_\kappa : \{0,1\}^* \rightarrow Z_q$ be cryptographic hash functions that are modelled as random oracles (RO). Roughly speaking, the JKEA assumption says that, given $X = xP$ for $x \leftarrow Z_q$, the ability of an efficient algorithm $\mathcal{A}$ to output $\{(Y_1, m_1), \cdots, (Y_\kappa, m_\kappa), Z\}$ such that $Z = x(\sum_{i=1}^{\kappa} e_i Y_i)$, where $Y_i \in E(F)$ and $m_i \in \{0,1\}^*$ and $e_i = H_i(Y_i, m_i)$ for $1 \leq i \leq \kappa$, implies knowing $(y_1, \cdots, y_\kappa)$ simultaneously, where y_i is the discrete logarithm of Y_i. Here, "knowing" implies that $(y_1, \cdots, y_\kappa)$ can be efficiently extracted by an extractor algorithm $\mathcal{E}$ from the input and the random tape of $\mathcal{A}$. The JKEA assumption is justified in [69] by the fact that, assuming H_i's are random oracles, no efficient algorithm can make the values in $\{e_1 Y_1, \cdots, e_\kappa Y_\kappa\}$ correlated. That is, no matter how the PPT algorithm $\mathcal{A}$ does, the values $\{H_1(Y_1, m_1)y_1, \cdots, H_\kappa(Y_\kappa, m_\kappa)y_\kappa\}$ are computationally independent as defined in [69].

The JKEA assumption implies the following weaker assumption, referred to as *explicit* knowledge-of-exponent assumption (EKEA). Specifically, the ability of outputting $\{(Y_1, m_1), \cdots, (Y_\kappa, m_\kappa), z\}$, satisfying $z \in Z_q$ and $zP = \sum_{i=1}^{\kappa} e_i Y_i$, implies knowing $(y_1, \cdots, y_\kappa)$ simultaneously. That is, $(y_1, \cdots, y_\kappa)$ can be efficiently extracted. Unlike the JKEA assumption where the algorithm $\mathcal{A}$ only outputs $CDH(X, \sum_{i=1}^{\kappa} e_i Y_i) = zX$, here $\mathcal{A}$ *explicitly* outputs the discrete logarithm $z = \log_P(\sum_{i=1}^{\kappa} e_i Y_i)$. Clearly, the EKEA assumption is implied by, and weaker than, the JKEA assumption.

Definition 5.1 (non-malleable discrete logarithm (NMDL) assumption). Let $G = (E(F_p), P, q)$ define a cyclic group over $E(F_p)$ generated by P of order q, where p and q are prime numbers, and $l = \lceil \log q \rceil$ be the security parameter. Let $H_1, \cdots, H_\kappa : \{0,1\}^* \rightarrow Z_q^*$ be cryptographic hash functions, which may not be distinct. On input (G, X) where $X = xP$ for $x \leftarrow Z_q^*$, a PPT algorithm $\mathcal{A}$ (called an NMDL-solver) succeeds in solving the NMDL problem, if it could output $\{(b_1, Y_1, m_1) \cdots, (b_\kappa, Y_\kappa, m_\kappa), z\}$, satisfying:

- $z \in Z_q$, and for any i, $1 \leq i \leq \kappa$, $Y_i \in G$, $m_i \in \{0,1\}^*$ that can be the empty string, and $b_i \in \{0,1\}$.
- For any $1 \leq i \neq j \leq \kappa$, it holds that $(Y_i, m_i) \neq (Y_j, m_j)$. But it might be the case that $Y_i = Y_j$ or $m_i = m_j$.
- $X \in \{Y_1, \cdots, Y_\kappa\}$, and $zP = \sum_{i=1}^{\kappa} (-1)^{b_i} e_i Y_i$ where $e_i = H_i(Y_i, m_i)$.

Then, the NMDL assumption says that, for any PPT algorithm $\mathcal{A}$, the probability that it succeeds in solving the NMDL problem is negligible in l. The probability is taken over the random coins used to generate (G, x), the random coins used by $\mathcal{A}$ (and the choices of the random functions $H_1, \cdots, H_\kappa$ in the random oracle model).

It is clear that the NMDL assumption is implied by the standard discrete logarithm assumption and the above (non-black-box) EKEA assumption. Note also that the NMDL assumption is itself black-box in nature. Below, we further justify this assumption by proving that it holds in the generic group and random oracle model [13, 15, 62, 64], where $H_1, \cdots, H_\kappa$ are assumed to be random oracles (RO) [12].

Briefly speaking, an algorithm is generic if it does not use the encoding of the group elements. It can only use group elements for group operations and relation verifications. There are many groups

```
$\textbf{KeyGen}(1^l)$
  $x \leftarrow Z_q^*$
  $X := xP$
  return $(x, X)$
```

```
$\textbf{Sign}(X, x, m)$
  $r \leftarrow Z_q^*$
  $A := rP$
  $d := H(A)$
  $e := H(X, m)$
  $z := rd - ex \mod q$
  return $\sigma = (d, z)$
```

```
$\textbf{Verify}(X, m, \sigma = (d, z))$
  $e := H(X, m)$
  $A := zd^{-1}P + ed^{-1}X$
  if $H(A) \neq d$
    return $REJECT$
  else
    return $ACCEPT$
```

```
$\textbf{Agg}(\{(X_1, m_1, \sigma_1), ..., (X_n, m_n, \sigma_n)\})$
  $\hat{T} := \emptyset, \hat{A} := \emptyset, z := 0$
  for $i = 1$ to $n$
  if $Verify(X_i, m_i, \sigma_i) = ACCEPT \land (X_i, m_i) \notin \hat{T} \land A_i \notin \hat{A}$
    $\hat{T} := \hat{T} \cup \{(X_i, m_i)\}$
    $\hat{A} := \hat{A} \cup \{A_i\}$
    $z := z + z_i \mod q$
  return $(\hat{T}, \hat{A}, z)$
```

```
$\textbf{AggVerify}(\hat{T}, \hat{A}, z)$
  if elements in $\hat{T}$ are not distinct
    return $REJECT$
  if elements in $\hat{A}$ are not distinct
    return $REJECT$
  if $|\hat{T}| \neq |\hat{A}|$
    return $REJECT$
  $n' := |\hat{T}| = |\hat{A}|$
  for $j = 1$ to $n'$
    $d_j := H(A_j), e_j := H(X_j, m_j)$
  if $(zP + \sum_{j=1}^{n'}(e_j X_j - d_j A_j)) \neq \infty$
    return $REJECT$
  return $ACCEPT$
```

Table 2: Aggregate Γ-signature

for which the fastest DL solver algorithms are generic. For example, general elliptic curves; general hyper-elliptic curves of genus 2; and subgroups of prime order q in Z_p^* when $(p-1)/q$ is so large that sieving methods are inefficient [63]. For presentation simplicity, in the following analysis we use Maurer's generic group model [45] that is actually equivalent to Shoup's model [37, 66].

Theorem 5.2. *For an NMDL-solver algorithm that runs τ generic steps and makes ϱ RO-queries, its success probability is upper bounded by $\frac{\tau^2+\varrho^2}{q-1}$ in the generic group and random oracle model.*

Proof. In Maurer's generic group model for solving the NMDL problem, the generic group oracle (GG-oracle) O originally keeps two internal states $(1, x)$ in a list L, where $x \leftarrow Z_q^*$. For presentation simplicity, we denote by $L[i]$ the value stored in the i-th entry of L, where $1 \leq i \leq \tau$, and we assume $L[1] = 1$ and $L[2] = x$. The generic NMDL-solver algorithm $\mathcal{A}$ is given the indices of $(1, x)$ in L, i.e., $(1, 2)$, and has black-box access to the GG-oracle O and a random oracle (RO) $\mathcal{H} : \{0, 1\}^* \rightarrow Z_q^*$. Here, for presentation simplicity, we use a single random oracle $\mathcal{H}$ to represent $\{H_1, \cdots, H_\kappa\}$.

For the i-th GG-oracle access corresponding to a group operation, $1 \leq i \leq \tau$, the value computed by the GG-oracle O can be viewed as a linear polynomial of the form $F_i(x) = a_i x + b_i \mod q$, where $a_i, b_i \in Z_q$ are determined by previous GG-oracle accesses. The value F_i is not returned to $\mathcal{A}$ directly, but is stored into a position in the internal list L where the position index for storing F_i is indicated by $\mathcal{A}$. $\mathcal{A}$ is always given the ability of verifying equality relation, by which $\mathcal{A}$ queries O with (i, j) and gets result whether $L[i] = L[j]$ or not. For the k-th RO query, $1 \leq k \leq \varrho$, the algorithm A queries the random oracle $\mathcal{H}$ with (t_k, m_k), where $m_k \in \{0, 1\}^*$ and $1 \leq t_k \leq \tau$ represents the index of the component $F_{t_k}(x)P$ that is actually unknown to $\mathcal{A}$ in the generic group model. Upon the k-th RO-query (t_k, m_k), the random oracle $\mathcal{H}$ works as follows: (1) if $L(t_k)$ is undefined, it returns $\perp$ indicating invalid RO-query;[3] (2) if $\mathcal{H}(t_k, m_k)$ has been defined, it returns what already defined; (3) otherwise, it defines and returns a value taken uniformly at random from Z_q^* as $\mathcal{H}(t_k, m_k)$. Finally, $\mathcal{A}$ outputs $\{(b_1, i_1, m_{i_1}), \cdots, (b_\kappa, i_\kappa, m_{i_\kappa}), z\}$, and succeeds on the following conditions:

- $z \in Z_q$, $b_\alpha \in \{0, 1\}$ and $m_\alpha \in \{0, 1\}^*$ where $1 \leq \alpha \leq \kappa$, and $1 \leq i_\beta \leq \tau$ for $1 \leq \beta \leq \kappa$. Here, i_β is the index of $F_{i_\beta}(x)P$.
- For any $1 \leq \alpha \neq \beta \leq \kappa$, it holds that $(i_\alpha, m_\alpha) \neq (i_\beta, m_\beta)$.
- $2 \in \{i_1, \cdots, i_\kappa\}$ where the index 2 represents the input $X = xP$ to the NMDL-solver $\mathcal{A}$ in the generic group model, and $z = \sum_{\alpha=1}^{\kappa}(-1)^{b_\alpha} e_{i_\alpha} F_{i_\alpha}(x) \mod q$ where $e_{i_\alpha} = \mathcal{H}(i_\alpha, m_{i_\alpha}) \in Z_q^*$.

As discussed in [45], in this generic group model we only need to consider non-adaptive adversaries, and there are only three approaches for $\mathcal{A}$ to succeed in the generic group model.

- Simply guessing x, which succeeds with probability $\frac{1}{q-1}$.
- Another approach is to cause two different F_i and F_j to collide, $1 \leq i, j \leq \tau$, in the sense that $a_i x + b_i = a_j x + b_j$ where $(a_i, b_i) \neq (a_j, b_j)$. In other words, $(a_i - a_j)x + (b_i - b_j) = 0$. By Schwartz-Shoup lemma [45, 46, 65, 66], this event can occur with probability at most $C_\tau^2 \frac{1}{q-1} = \frac{\tau(\tau-1)}{q-1}$.
- The third approach for $\mathcal{A}$ to succeed is to output $\{(\hat{b}_1, \hat{i}_1, \hat{m}_{\hat{i}_1}), \cdots, (\hat{b}_\gamma, \hat{i}_\gamma, \hat{m}_{\hat{i}_\gamma}), \hat{z}\}$ such that $\hat{z} = \sum_{\alpha=1}^{\gamma}(-1)^{\hat{b}_\alpha} e_{\hat{i}_\alpha} F_{\hat{i}_\alpha}(x) \mod q$, where $\gamma > 1$ and $e_{\hat{i}_\alpha} = \mathcal{H}(\hat{i}_\alpha, \hat{m}_{\hat{i}_\alpha}) \in Z_q^*$. The observation here is that, for any tuple $\{(\hat{b}_1, \hat{i}_1, \hat{m}_{\hat{i}_1}), \cdots, (\hat{b}_\gamma, \hat{i}_\gamma, \hat{m}_{\hat{i}_\gamma}), \hat{z}\}$, the probability that $\hat{z} = \sum_{\alpha=1}^{\gamma}(-1)^{\hat{b}_\alpha} e_{\hat{i}_\alpha} F_{\hat{i}_\alpha}(x) \mod q$ is at

[3] In this case, $\mathcal{H}(t_k, m_k)$ remains undefined. This is to ensure the independence between F_{t_k} and $\mathcal{H}(t_k, \cdot)$.

most $\frac{1}{q-1}$ in the random oracle model. Then, by the birthday paradigm, the probability that $\mathcal{A}$ succeeds with this approach is at most $\frac{\varrho^2}{q-1}$, where ϱ is the number of queries made by $\mathcal{A}$ to the random oracle $\mathcal{H}$.

Note that $\frac{1}{q-1} + C_\tau^2 \frac{1}{q-1} + \frac{\varrho^2}{q-1} < \frac{\tau^2+\varrho^2}{q-1}$. □

5.2 Security Analysis

Theorem 5.3. *The aggregate* Γ*-signature scheme presented in Table 2 is secure (in accordance with Definition 3.1) under the NMDL assumption in the random oracle model.*

Proof. According to the security definition of aggregate signature presented in Section 3, supposing there exists a PPT forger $\mathcal{A}$ who breaks the security of the aggregate Γ-signature with non-negligible probability, we present another PPT algorithm $\mathcal{B}$ who can solve the NMDL problem also with non-negligible probability. Denote by $(X = xP, x)$ the public and secret key pair of the target honest user, where $x \leftarrow Z_q^*$. The algorithm $\mathcal{B}$ takes (G, X) as input (where G is the underlying cyclic group defined in the elliptic curve), runs $\mathcal{A}$ as a subroutine, and works as follows.

Whenever $\mathcal{A}$ asks the target user to sign a message m, $\mathcal{B}$ answers the signing query by running the Γ-signature simulator $\mathcal{S}$ as described in Corollary 2.1, where the simulation is statistically indistinguishable from what $\mathcal{A}$ gets in reality. Finally, suppose that $\mathcal{A}$ outputs a valid aggregate Γ-signature denoted $(\hat{T}, \hat{A}, z)$, where $\hat{T} = \{(X_1, m_1), \cdots, (X_{n'}, m_{n'})\}$, $\hat{A} = \{A_1, \cdots, A_{n'}\}$. Assume that there are n'' distinct elements $\bar{X} = \{X_{i_1}, \cdots, X_{i_{n''}}\}$ in $\hat{X} = \{X_1, \cdots, X_{n'}\}$, where X_{i_j} appears t_j times in $\hat{X}$ and $\sum_{j=1}^{n''} t_j = n'$. For each j, $1 \le j \le n''$, denote by $I_j = \{j_1, \cdots, j_{t_j}\}$ the set of indices that X_{i_j} appears in $\hat{X}$ where $1 \le j_\alpha \le n'$ for $1 \le \alpha \le t_j$; specifically, $X_{i_j} = X_{j_1} = \cdots = X_{j_{t_j}}$. $\mathcal{B}$ outputs $\{\bar{T}, \bar{A}, z\}$, which are specified below:

- $\bar{T} = \{(b_1, X_{i_1}, m_1), \cdots, (b_{n''}, X_{i_{n''}}, m_{n''})\}$, where for each j, $1 \le j \le n''$, $b_j = -1$ and $m_j = m_{j_1} || \cdots || m_{j_{k_j}}$.
- $\bar{A} = \{(b'_1, A_1, \lambda) \cdots, (b'_{n'}, A_{n'}, \lambda)\}$, where for each i, $1 \le i \le n'$, $b'_i = 1$, and λ represents the empty string.

According to the security analysis of Γ-signature in [70], what seen by $\mathcal{A}$ under the run of $\mathcal{B}$ is statistically indistinguishable from what seen in reality. Thus, with also non-negligible probability, $\mathcal{A}$ will output a *valid* aggregate Γ-signature $(\hat{T}, \hat{A}, z)$ under the simulation of $\mathcal{B}$. Consequently, $\mathcal{B}$ outputs $(\bar{T}, \bar{A}, z)$ with the same probability. Define $H' : G \times (\{0,1\}^*)^\beta \rightarrow Z_q$ as follows: $H'(X, m_1, \cdots, m_\beta) = H(X, m_1) + \cdots + H(X, m_\beta) \mod q$ for any β, $1 \le \beta \le n'$. Assuming $H : \{0,1\}^* \rightarrow Z_q$ is a random oracle, so is H'. Finally, we show that the output $\{\bar{T}, \bar{A}, z\}$ by $\mathcal{B}$ is a correct solution to the NMDL problem, by the following observations:

- All the tuples in $\bar{T} \bigcup \bar{A}$ are distinct. This is from the facts that: (1) the tuples in $\bar{T}$ are distinct and $m_j \ne \lambda$, $1 \le j \le n''$; (2) the tuples in $\bar{A}$ are also distinct with the same empty string as the third element in each tuple.
- As we assume the aggregate signature $(\hat{T}, \hat{A}, z)$ output by $\mathcal{A}$ is valid, we have that $X \in \bar{X} = \{X_{i_1}, \cdots, X_{i_{n''}}\}$, and $zP = \sum_{i=1}^{n'} d_i A_i - \sum_{j=1}^{n''} e'_j X_{i_j}$, where $d_i = H(A_i)$ and $e'_j = H'(X_{i_j}, m_j)$.

6 APPLICATIONS TO BITCOIN

In this section, we describe a Merkle-Patricia tree (MPT) aided implementation of our aggregate signature scheme, and specify its applications to Bitcoin. The goal is to maximize performance and compatibility with the existing Bitcoin system, with the least modifications that are inherent in deploying aggregate signatures. Our modifications involve: txid, unlocking script, locking script, Merkle tree, block construction, block mining and block verification. For presentation simplicity, we describe our implementation for a hard-fork of Bitcoin, in a self-contained manner for ease of reading.

6.1 Inheritances: Keys, Addresses and Network

Bitcoin uses a specific elliptic curve, as defined in a standard called secp256k1, established by NIST. Our aggregate Γ-signature scheme also works on the secp256k1 curve. As for new key pair generation, algorithm $KeyGen(1^l)$ is the same as in the existing Bitcoin system.

As for Bitcoin addresses, we inherit the existing design in Bitcoin. Specifically, this is the process of generating address from public key through the use of one-way hash algorithms SHA256 and RIPEMD160,

A=RIPEMD160(SHA256(X)),

where X is the public key and A is the Bitcoin address. The above address is called P2PKH address. There is another type of address called P2SH address, which is generated by the following equation:

A=RIPEMD160(SHA256(script)).

We also use the Base58 [3] and Base58Check [2, 7] formats for unambiguously and compactly encoding Bitcoin data such as addresses, etc.

We adopt the existing Bitcoin network which is structured as a peer-to-peer (P2P) network on top of the internet. And the Bitcoin network refers to the collection of nodes running the Bitcoin protocol. When a peer receives data, it will broadcast the data to its neighbouring peers after some necessary verification. With the usage of P2P network, in a very short period of time, the data such as transactions and blocks can be efficiently spread all over the network.

6.2 Transactions

Transactions are the most important part of the Bitcoin system. Everything else in Bitcoin is designed to ensure that transactions can be created, propagated on P2P network, validated, and finally added to the global ledger of transactions (i.e., the blockchain).

The Bitcoin transaction consists of fields such as version, in-counter, inputs list, out-counter, outputs list and locktime, which is shown in Table 3.

Within the inputs list field of transaction, it consists of

- **txid**: a pointer to the transaction containing the unspent transaction output (UTXO).
- **vout**: the index number of the UTXO to be spent.
- **unlocking script**: a script that fulfills the conditions of the UTXO locking script.
- **sequence**: the block number where the UTXO is recorded in the blockchain.

In the Bitcoin system, txid is the double SHA256 hash of the transaction, including the witness (i.e., the associated signature).

Field	Description	Size
version	Transaction version number	4 bytes
in-counter	Counter of inputs	1-9 bytes
inputs-list	List of transaction inputs	variable
out-counter	Counter of outputs	1-9 bytes
outputs-list	List of transaction outputs	variable
locktime	Earliest time that a transaction is valid	4 bytes

Table 3: Structure of Bitcoin transaction

It is inherently impossible to retrieve the txid whenever aggregate signature is used, where multiple individual signatures are replaced with the aggregate signature in the block. So, for AS-based implementations, we modify the txid to be the double SHA256 hash of the transaction without witness. Note that tampering with the witness data is the source for launching transaction malleability attacks [20]. Removing it from the hash input in generating txid also removes the opportunity for *transaction malleability attacks*. This can also greatly improve the implementations for many other protocols, such as payment channels, chained transactions, and lightning networks.

Unlocking script of P2PKH is in the format of <sig><PubK>, where PubK is a public key and sig is a signature signed by the private key corresponding to PubK; The unlocking script of P2SH has a basic format of <sig I><sig J>, mainly for multi-signature. In our modifications, the sig is generated by the $Sign(X, x, m)$ of Γ-signature (where m is part of a transaction defined by SIGHASH flag), which replaces the existing EC-DSA signature.

Within the outputs list field of transaction, it consists of (1) **value** which is an amount of Bitcoin; and (2) **locking script** which is a cryptographic puzzle that determines the conditions required to spend the output. As for operations OP_CHECKSIG and OP_CHECKMULTISIG among locking script of P2PKH and P2SH, the EC-DSA verification procedure is replaced by running $Verify(X, m, \sigma = (d, z))$ of Γ-signature.

6.3 Block

A block is a container data structure that collects transactions for inclusion in the public ledger, the blockchain. The block consists of a header, containing metadata, followed by a long list of transactions, which is shown in Table 4.

Field	Description	Size
magic-no	Value always $0xD9B4BEF9$	4 bytes
blocksize	Number of bytes following up to end of block	4 bytes
blockheader	Consists of 6 items	80 bytes
tx-counter	Counter of transactions	1-9 bytes
transactions	List of transactions	variable

Table 4: Structure of Bitcoin block

Each block is identified by a hash which is generated by running the SHA256 cryptographic hash algorithm twice on the block header. The size of block header is 80-bytes, and its structure is shown in Table 5.

Field	Description	Size
version	Block version number	4 bytes
hashPrevBlock	Hash of the previous block header	32 bytes
hashMerkleRoot	Hash of Merkle tree root in the block	32 bytes
timestamp	Current timestamp as seconds	4 bytes
bits	Current target in compact format	4 bytes
nonce	32-bit number	4 bytes

Table 5: Structure of Bitcoin blockheader

Every block in blockchain contains a summary of all the transactions using a Merkle tree. A Merkle tree, also known as a binary hash tree, is a data structure used for efficiently summarizing and verifying the integrity of large sets of data. In our modifications, we build Merkle tree with our modified txid which is the double SHA256 hash of the transaction without witness.

In the existing Bitcoin system, after validating transactions a miner will add them to the memory pool or transaction pool where transactions await until they can be included (mined) into a block. We adopt Merkle-Patricia tree (MPT) [54] to play the role of memory pool and to perform duplication check, as the elements in $\hat{T}$ and $\hat{A}$ in our aggregate Γ-signature are required to be distinct. MPT can provide a cryptographically authenticated data structure that can be used to store $(key, value)$ pairs, and enjoys a faster speed both in element searching and in outputting ordered elements. The algorithm Agg in our aggregate Γ-signature can be implemented with MPT as follows.

- Initialize two empty MPT instances $MPT_{\hat{A}}$ and $MPT_{\hat{T}}$, where $MPT_{\hat{A}}$ (resp., $MPT_{\hat{T}}$) is for the set of $\hat{A}$ (resp., $\hat{T}$).
- Traverse the received transactions and do the following. For every transaction input, extract the public key X_i and the signature $\sigma_i = (d_i, z_i)$ in the unlocking script, and m_i that is the specific part of a transaction defined by SIGHASH flag; Then, calculate A_i from σ_i, and search in MPT_A, MPT_T to check whether there already exists A_i or (X_i, m_i); Finally, verify (X_i, m_i, σ_i) with our $Verify$ algorithm.
- If there already exists A_i or (X_i, m_i), or $Verify$ algorithm outputs $REJECT$, drop the current transaction, and loop to the next transaction.
- Insert A_i and (X_i, m_i) to MPT_A and MPT_T respectively, and set $z := z + z_i \mod q$.
- When all the transactions are traversed, output the ordered list of A_i's as $\hat{A}$, the ordered list of (X_i, m_i) as $\hat{T}$, and a number $z \in Z_q$.

Now, we pay attention to P2SH unlocking script multi-signature, where N public keys are recorded in the script and at least M of them must provide signatures to unlock the funds. In order to aggregate the multi-signature, a Bitcoin node should extract each tuple (X_j, m, σ_j) from the M provided signatures on the same message m, and deals with it like a normal transaction input.

After collecting enough transactions, the miner constructs a candidate block, with the only witness of aggregate signature $(\hat{A}, z)$

being placed at the end of block as specified by segregated witness (SegWit).[4] This way, our result inherits all the advantages of SegWitness, besides enjoying a more compact witness.

When a miner finds a solution nonce (that is inserted into the block header) such that the block header hash is less than the target, the miner transmits the candidate block to all its peers immediately. By the consensus mechanism of Bitcoin, every node independently validates the new block before propagating it to its peers, which ensures that only valid blocks are propagated on the network. Instead of individually validating all the transactions within the block, with our modifications, each node only needs to simply verify one aggregate signature with algorithm $AggVerify(\hat{T}, \hat{A}, z)$, as follows.

- Note that both $\hat{T}$ and $\hat{A}$ are ordered. In order to ensure the elements within are distinct, just traverse the lists $\hat{T}$ and $\hat{A}$ to confirm that every two adjacent elements are different and are monotonically incremented.
- If the elements are not distinct in the above step, abort and output $REJECT$. Otherwise, continue with the next procedure.
- Execute the aggregate signature validation, and output $ACCEPT$ if the verification is successful. Otherwise, output $REJECT$.

7 CONCLUSION AND FUTURE WORK

In this work, we present the first aggregate signature (AS) scheme from general elliptic curve groups without bilinear maps. Compared to a list of individual signatures (on potentially pairwise distinct messages), the storage volume of signatures reduces about 49.8% and the signature verification time can even reduce about 72% with the proposed AS scheme. Its provable security is based on a new assumption, named non-malleable discrete logarithm (NMDL), which is proved in the generic group and random oracle model (and is also implied by the DL assumption and a weaker non-black-box assumption). We suggest the NMDL assumption should be of independent interest. Finally, we specify in detail the application of the proposed AS scheme to Bitcoin, with the goal of maximizing performance and compatibility. Towards that, we adopt a Merkle-Patricia tree based implementation of our AS scheme. Besides security inherited from Bitcoin, the AS-aided system is also more friendly to segregated witness, and provides better protection against transaction malleability attacks.

Though using both generic group model and random oracle model is not rare (particularly for arguing security of practical cryptographic schemes, e.g. [13, 15, 62, 64]), it is interesting to investigate whether practical AS schemes from general groups can be built with provable security only in the random oracle model. Note that our aggregate signature only about halves the bandwidth or storage volume of signatures. Studying the (im)possibility of constant-size AS from general groups is an important question for future research, on which we are inclined to a theoretical impossibility result (at least for the case of black-box security reduction).

ACKNOWLEDGMENTS

We are grateful to Xing Chang, Leixiao Cheng, Boru Gong, Xingzhong Huang, Bao Li, Wei Yu and Andrew C. Yao for many helpful discussions and assistance.

This work is supported in part by National Key Research and Development Program of China under Grant No. 2017YFB0802000, National Natural Science Foundation of China under Grant Nos. 61472084 and U1536205, Shanghai Innovation Action Project under Grant No. 16DZ1100200, Shanghai Science and Technology Development Funds under Grant No. 16JC1400801, and Shandong Provincial Key Research and Development Program of China under Grant Nos. 2017CXG0701 and 2018CXGC0701.

REFERENCES

[1] M. Abe and S. Fehr. Perfect NIZK with Adaptive Soundness. *TCC* 2007: 118-136.
[2] A. M. Antonopoulos. Mastering Bitcoin. Available at https://github.com/bitcoinbook/bitcoinbook
[3] A. M. Antonopoulos. Mastering Bitcoin. Section: Base58. Available at https://github.com/bitcoinbook/bitcoinbook
[4] A. Bagherzandi, J.H. Cheon, and S. Jarecki. Multisignatures Secure Under the Discrete Logarithm Assumption and a Generalized Forking Lemma. *ACM Conference on Computer and Communications Security* 2008: 449–458.
[5] B. Barak. How to Go Beyond the Black-Box Simulation Barrier. *FOCS* 2001: 106-115.
[6] R. Barbulescu and S. Duquesne. Updating key size estimations for pairings. *Journal of Cryptology*, 2018: 1-39.
[7] Base58Check Encoding. Available at https://en.bitcoin.it/wiki/Base58Check_encoding
[8] M. Bellare, C. Namprempre and G. Neven. Unrestricted Aggregate Signatures. *ICALP* 2007: 411-422.
[9] M. Bellare and G. Neven. Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma. *ACM Conference on Computer and Communications Security* 2006: 390-399.
[10] M. Bellare and A. Palacio. Towards Plaintext-Aware Public-Key Encryption without Random Oracles. *ASIACRYPT* 2004: 48-62.
[11] M. Bellare and A. Palacio. The Knowledge-of-Exponent Assumptions and 3-Round Zero-Knowledge Protocols. *CRYPTO* 2004: 273-289.
[12] M. Bellare and P. Rogaway. Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. *ACM CCS* 1993: 62–73.
[13] Bethencourt, J., Sahai, A., Waters, B. Ciphertext-Policy Attribute-Based Encryption. *IEEE Symposium on Security and Privacy (S&P)* 2007, 321-334.
[14] A. Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. *PKC* 2003, LNCS 2567, Springer-Verlag.
[15] A. Boldyreva, C. Gentry, A. O'Neill and D. H. Yum. Ordered Multisignatures and Identity-Based Sequential Aggregate Signatures, with Applications to Secure Routing. *CCS* 2007: 276-285.
[16] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again. *ITCS* 2012: 326-349.
[17] D. Boneh, M. Drijvers, and G. Neven. Compact Multi-Signatures for Smaller Blockchains. *ASIACRYPT* 2018, to appear.
[18] D. Boneh, C. Gentry, B. Lynn and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. *EUROCRYPT* 2003: 416-432.
[19] D. Boneh, B. Lynn and H. Shacham. Short Signatures from the Weil Pairing. *ASIACRYPT* 2001: 514-532.
[20] D. Bradbury. What the 'Bitcoin Bug' Means: A Guide to Transaction Malleability. Available at https://www.coindesk.com/bitcoin-bug-guide-transaction-malleability
[21] R. Canetti and R. R. Dakdouk. Extractable Perfectly One-Way Functions. *ICALP* (2) 2008: 449-460.
[22] R. Canetti and R. R. Dakdouk. Towards a Theory of Extractable Functions. *TCC* 2009: 595-613.
[23] C. Research. SEC 2: Recommended Elliptic Curve Domain Parameters 2010. Available at http://www.secg.org/sec2-v2.pdf
[24] I. Damgård. Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks. *CRYPTO* 1991: 445-456.
[25] I. Damgård, S. Faust and C. Hazay. Secure Two-Party Computation with Low Communication. *TCC* 2012: 54-74.
[26] A. W. Dent. The Cramer-Shoup Encryption Scheme is Plaintext Aware in the Standard Model. *EUROCRYPT* 2006: 289-307.
[27] V. S. Dimitrov, G. A. Jullien, and W. C. Miller. Complexity and Fast Algorithms for Multiexponentiations. *IEEE Trans. Computers* (2) 2000: 141-147.

[4] Segregated witness is an architectural change to Bitcoin, which aims to move the witness data from the field of scriptSig (unlocking script) in a transaction into a separate witness data structure.

[28] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. *CRYPTO* 1986: 186-194.

[29] R. Gennaro, H. Krawczyk, and T. Rabin. Okamoto-Tanaka Revisited: Fully Authenticated Diffie-Hellman with Minimal Overhead. *ACNS* 2010: 309-328.

[30] S. Goldwasser, H. Lin, and A. Rubinstein. Delegation of Computation without Rejection Problem from Designated Verifier CS-Proofs. *IACR Cryptology ePrint Archive* 2011: 456.

[31] D. M. Gordon. A Survey of Fast Exponentiation Methods. *J. Algorithms* 27(1) 1998: 129-146.

[32] J. Groth. Short Pairing-Based Non-Interactive Zero-Knowledge Arguments. *ASIACRYPT* 2010: 321-340.

[33] S. Hada and T. Tanaka. On the Existence of 3-Round Zero-Knowledge Protocols. *CRYPTO* 1998: 408-423.

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

[35] S. Hohenberger, B. Waters. Synchronized Aggregate Signatures from the RSA Assumption. *EUROCRYPT* 2018: 197-229.

[36] K. Itakura and K. Nakamura. A Public-Key Cryptosystem Suitable for Digital Multisignatures. *NEC Research & Development*, 71:1–8, 1983.

[37] T. Jager and J. Schwenk. On the Equivalence of Generic Group Models. *ProvSec* 2008: 200-209.

[38] D. Johnson, A. Menezes and S. Vanstone. The Elliptic Curve Digital Signature Algorithm (EC-DSA). *Int. J. Inf. Sec* 1(1) 2001: 36-63.

[39] H. Krawczyk. HMQV: A High-Performance Secure Diffie-Hellman Protocol. *CRYPTO* 2005: 546-566.

[40] C.M. Li, T. Hwang, and N.Y. Lee. Threshold Multisignature Schemes where Suspected Forgery Implies Traceability of Adversarial Shareholders. *EUROCRYPT* 1994, LNCS 950, Springer-Verlag.

[41] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. Sequential Aggregate Signatures and Multisignatures without Random Oracles. *EUROCRYPT* 2006, LNCS 4004, Springer-Verlag.

[42] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential Aggregate Signatures from Trapdoor Permutations. *EUROCRYPT* 2004, LNCS 3027, Springer-Verlag.

[43] C. Ma, J. Weng, Y. Li and R. H. Deng. Efficient Discrete Logarithm Based Multi-Signature Scheme in the Plain Public Key Model. *Codes Cryptography* 54(2) 2010: 121-133.

[44] W. Mao. Modern Cryptography: Theory and Practice. *CRC* 2004.

[45] U. Maurer. Abstract Models of Computation in Cryptography. *IMA Cryptography and Coding* 2005: 1-12.

[46] U. Maurer and S. Wolf. Lower Bounds on Generic Algorithms in Groups. *EUROCRYPT* 1998: 72-84.

[47] G. Maxwell. Signature Aggregation for Improved Scalablity. Available at https://bitcointalk.org/index.php?topic=1377298.0

[48] G. Maxwell, A. Poelstra, Y. Seurin and P. Wuille. Simple Schnorr Multi-Signatures with Applications to Bitcoin. *IACR Cryptology ePrint Archive* 2018: 68.

[49] S. Micali, K. Ohta, and L. Reyzin. Accountable-Subgroup Multisignatures. *ACM CCS* 2001, ACM Press.

[50] T. Mie. Polylogarithmic Two-Round Argument Systems. *J. Mathematical Cryptology* 2(4) 2008: 343-363.

[51] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008 Available at http://bitcoin.org/bitcoin.pdf

[52] K. Ohta and T. Okamoto. A Ddigital Multisignature Scheme Based on the Fiat-Shamir Scheme. *ASIACRYPT* 1991, LNCS 739, Springer-Verlag.

[53] L. Parker. Bitcoin 'Spam Attack' Stressed Network for at least 18 Months, Claims Software Developer. Available at https://bravenewcoin.com/news/bitcoin-spam-attack-stressed-network-for-at-least-18-months-claims-software-developer/

[54] Patricia Tree. Available at https://github.com/ethereum/wiki/wiki/Patricia-Tree

[55] D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, 13(2) 2000: 36-396.

[56] M. D. Raimondo and R. Gennaro. New Approaches for Deniable Authentication. *ACM Conference on Computer and Communications Security* 2005: 112-121.

[57] M. D. Raimondo, R. Gennaro, and H. Krawczyk. Deniable Authentication and Key Exchange. *ACM Conference on Computer and Communications Security* 2006: 400-409.

[58] T. Ristenpart and S. Yilek. The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks. *EUROCRYPT* 2007: 228-245.

[59] E. B. Sasson, A. Chiesay, C. Garmanz, M. Greenz, I. Miersz, E. Tromerx and M. Virza. Zerocash: Decentralized Anonymous Payments from Bitcoin. *IEEE Symposium on Security and Privacy* 2014: 459-474.

[60] E. B. Sasson, A. Chiesa, E. Tromer and M. Virz. Succinct Non-Interactive Zero Knowledge for a Von Neumann Architecture. *USENIX Security* 2014: 781-796.

[61] C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. *CRYPTO* 1989: 239-252.

[62] C. P. Schnorr. Security of Blind Discrete Log Signatures against Interactive Attacks. *ICICS* 2001: 1-12.

[63] C. P. Schnorr. Small Generic Hardcore Subsets for the Discrete Logarithm. *Information processing Letters* 79(2): 93-98, 2001.

[64] C. P. Schnorr, M. Jakobsson. Security of Signed ElGamal Encryption. *ASIACRYPT* 2000: 73-89.

[65] J. T. Schwartz. Fast Probabilistic Algorithms for Verifications of Polynomial Identities. *Journal of the ACM*, 27(3): 701-717, 1980.

[66] V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. *EUROCRYPT* 1997: 256-266.

[67] A. V. Wirdum. Scriptless Scripts: How Bitcoin Can Support Smart Contracts Without Smart Contracts. Available at https://bitcoinmagazine.com/articles/scriptless-scripts-how-bitcoin-can-support-smart-contracts-without-smart-contracts/

[68] A. C.-C. Yao and Y. Zhao. Deniable Internet Key Exchange. *ACNS* 2010: 329-348.

[69] A. C.-C. Yao and Y. Zhao. OAKE: A New Family of Implicitly Authenticated Diffie-Hellman Protocols. *ACMCCS* 2013: 1113-1128. Full version available at https://eprint.iacr.org/2011/035

[70] A. C.-C. Yao and Y. Zhao. Online/Offline Signatures for Low-Power Devices. *IEEE Trans Information Forensics and Security* 8(2) 2013: 283-294.

[71] A. C.-C. Yao and Y. Zhao. Privacy-Preserving Authenticated Key-Exchange Over Internet. *IEEE Trans Information Forensics and Security* 9(1) 2014: 125-140.