Mathematics

AIMS Mathematics, 9(8): 20304–20325.
DOI: 10.3934/math.2024988
Received: 27 December 2023
Revised: 02 May 2024
Accepted: 23 May 2024
Published: 21 June 2024

https://www.aimspress.com/journal/Math

Research article

Nonce generation techniques in Schnorr multi-signatures: Exploring EdDSA-inspired approaches

Nawras H. Sabbry and Alla Levina*

Faculty of Computer Technologies and Informatics, ETU "LETI" University, St. Petersburg, Russia

*** Correspondence:** Email: ablevina@etu.ru.

Abstract: This paper proposes a deterministic nonce generation technique to address the catastrophic issues associated with nonce reuse in message signing and to enhance the efficiency of Schnorr multi-signature schemes. Additionally, this research aims to reduce computational complexity and bandwidth requirements in digital and multi-signature schemes while maintaining robust security against common attacks. The proposed method was inspired by the EdDSA approach. The methodology includes a comprehensive mathematical analysis of digital signature algorithms and a rigorous examination of their vulnerabilities to well-known cryptographic attacks. This analysis evaluates the effectiveness and robustness of the proposed nonce generation technique within the frameworks of the Schnorr digital signature and the two-round MuSig schemes. Techniques and tools employed in this research involve deterministically generating nonces by hashing the private key and subsequently hashing the result with the message. Furthermore, it is proposed to exclude the public nonce R from the challenge calculations and to allow signers to directly prove possession of their secret keys through the aggregated public key, thereby eliminating the need for non-interactive zero-knowledge (NIZK) proofs. The findings demonstrate significant reductions in computational complexity and operational requirements, thereby improving bandwidth efficiency and making this method well-suited for resource-constrained devices. The approach also exhibits strong resistance to various attacks, including nonce reuse, key cancellation, rogue keys, and virtual machine rewinding.

Keywords: Schnorr multi-signatures; MuSig schemes; EdDSA; nonce generation technique
Mathematics Subject Classification: 11T71, 94A60

1. Introduction

Digital signatures serve as a cornerstone of modern cryptographic practices, underpinning the security and trustworthiness of digital communications, financial transactions, and authentication processes.

The concept of digital signatures dates back to the 1970s, with the theoretical foundations laid by public key cryptography. Whitfield Diffie and Martin Hellman first introduced the idea in their seminal 1976 paper [1], paving the way for the development of various digital signature schemes. The first practical implementation, the RSA algorithm, was developed in 1977 by Rivest, Shamir, and Adleman [2]. It utilizes the computational difficulty of factoring large integers as the basis for security. This was followed by the introduction of the digital signature algorithm (DSA) in the early 1990s, which relies on the discrete logarithm problem [3]. Both schemes set the stage for subsequent advancements and adaptations in digital signature technology. The Schnorr signature scheme, introduced by Claus Schnorr [4], is distinguished by its simplicity and efficiency, particularly in terms of verification speed and shorter signatures. Later, the Edwards-curve digital signature algorithm (EdDSA) was developed to provide stronger security assurances and better performance using twisted Edwards curves [5]. Each of these developments has contributed to the robust framework within which digital signatures operate today, addressing various aspects of security and efficiency.

The evolution of digital signature schemes has been marked by a continuous quest for enhanced security, efficiency, and practical applicability. In this context, a nonce (number used once) is a random or pseudo-random number that must not be repeated with the same private key. However, the challenge of securely generating nonces (a critical component in the signature process) remains a pivotal concern that directly impacts the scheme's security and operational viability. For instance, in ECDSA (elliptic curve digital signature algorithm) and classic DSA, the uniqueness and secrecy of the nonce are crucial; if a nonce is revealed, it can lead to straightforward attacks that recover the private key. However, generating random nonces is fraught with challenges. The main vulnerability lies in the quality of the randomness. This issue was notably exploited in the PlayStation 3 firmware hack, where ECDSA nonces were predictably generated due to inadequate randomness, ultimately leading to the compromise of the private signing key.

In the digital signature schemes, the elliptic curve digital signature algorithm (ECDSA) [6] and the Edwards-curve digital signature algorithm (EdDSA) [7] have set benchmarks for security and performance. Particularly, EdDSA's approach to deterministic nonce generation has garnered attention for its ability to circumvent the pitfalls associated with random nonce generation, notably the risks of randomness failures and the subsequent exposure to attacks that could compromise the signer's private key. Therefore, the integration of deterministic nonce generation used in EdDSA's into Schnorr signatures presents a promising avenue for enhancing the robustness and efficiency of these protocols. Therefore, integrating deterministic nonce generation, as used in EdDSA, into Schnorr signatures presents a promising avenue for enhancing the robustness and efficiency of these protocols.

Previous studies have highlighted various approaches to nonce generation, emphasizing the risks associated with randomness failures. For instance, in their paper "Reusing Nonces in Schnorr Signatures (and keeping it secure...)" (2018) [8], Beunardeau et al. discuss the problem of nonce reuse in Schnorr signatures, which can reveal the secret signing key and compromise the security of the signature scheme. To overcome this issue, the authors propose a variant of the Schnorr signature scheme that allows for the reuse of nonces across multiple signatures while maintaining security. Their

variant uses a prime modulus p such that p-1 has several different factors q_i large enough to resist birthday attacks and mutualizes exponentiation efforts to achieve faster and more resource-efficient signatures. The authors also compare the performance of their scheme with classical Schnorr signatures and present several pre-computation techniques to speed up modular exponentiation, which is often the bottleneck in implementations of cryptographic systems.

It is worth noting that a lack of specific literature addressing the issue of nonce reuse attacks in EdDSA digital signatures was identified, largely because EdDSA is resistant to such attacks when the appropriate hashing function is chosen. Nonetheless, it is crucial to acknowledge that the security of any digital signature scheme, including EdDSA, is not impervious to potential vulnerabilities. For instance, fault attacks that exploit computational errors during the signature computation process [9], as well as other types of attacks that are unrelated to nonce reuse, could pose significant risks.

In their paper "Non-interactive Half-Aggregation of EdDSA and Schnorr Signatures" (2021) [10], Chalkias K., Garillot F., Kondi, Y., and Nikolaenko, V. offer a detailed investigation into non-interactive aggregation techniques for EdDSA and Schnorr signatures, focusing on optimizing signature aggregation for improved efficiency in cryptographic applications. The study delves into the intricacies of compact signature schemes, discussing the advantages and challenges of reducing signatures. By conducting thorough implementation and benchmarking analyses, the researchers demonstrate the practicality and performance of the proposed aggregation constructions. The results highlight the feasibility of the half-aggregation scheme and provide insights into the computational overhead required to achieve provable security guarantees in signature aggregation. The authors detail a methodology that not only enhances data compactness but also maintains the integrity and security of the signatures against various cryptographic attacks, making it highly relevant to investigations into the efficiency and security of Schnorr multi-signatures.

In another study, "Two-Round Stateless Deterministic Two-Party Schnorr Signatures from Pseudorandom Correlation Functions" (2023) [11], Kondi, Y., Orlandi, C., & Roy, L. introduced a novel protocol for two-party threshold Schnorr signatures that not only ensures stateless and deterministic signing but also addresses the inefficiencies found in previous threshold protocols. Utilizing pseudorandom correlation functions (PCFs), the study enhances both the security and efficiency of distributed Schnorr signing. These PCFs enable the distribution of signing nonces in a distributed, stateless, and deterministic manner, reducing the bandwidth cost per participant and making the system more suitable for practical applications.

The research provides a comprehensive analysis of various techniques for distributing Schnorr signatures, assessing key factors such as the number of rounds, bandwidth requirements, underlying assumptions, and security guarantees. The proposed protocols are designed to achieve both covert and full active security, offering robust protection against malicious actors under generic cryptographic assumptions. This advancement showcases significant improvements in the field of distributed signing protocols, particularly in how Schnorr signatures can be effectively and securely implemented in a threshold setting.

The research paper titled "MuSig-DN: Schnorr Multi-Signatures with Verifiably Deterministic Nonces" authored by Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille (2020) [12], provides significant insights into nonce generation and its implications for digital signature algorithms. The authors introduce a novel multi-signature scheme called MuSig-DN. This scheme utilizes deterministic nonces, which are generated in a manner that effectively safeguards against virtual machine rewinding attacks and nonce reuse attacks.

One of the pivotal challenges existing in MuSig-DN is the inherent complexity associated with deploying non-interactive zero-knowledge (NIZK) proofs to ascertain the deterministic generation of nonces. These proofs, while integral for securing the protocol against specific attack vectors and ensuring the determinism of nonce generation, introduce a notable layer of computational and conceptual complexity. Specifically, the protocol necessitates the construction and verification of NIZK proofs that each nonce was generated in accordance with the deterministic protocol. This requirement not only amplifies the computational overhead but also increases the intricacy of implementation, rendering the protocol less feasible for resource-constrained environments such as hardware wallets, which are prevalent in cryptocurrency applications.

The paper "MuSig2: Simple Two-Round Schnorr Multi-Signatures" (2023) [13], authored by Jonas Nick, Tim Ruffing, and Yannick Seurin, introduces a novel approach to multi-signatures by proposing the MuSig2 scheme, which optimizes the digital signature process. The research conducted in this paper focuses on enhancing the efficiency and security of multi-signature schemes, particularly in the context of Schnorr signatures. By leveraging the concept of using multiple nonces in the signing process, MuSig2 achieves significant improvements in security guarantees and performance metrics.

The key contributions of the paper include:

- Introducing MuSig2 as a two-round variant of the MuSig scheme eliminates the preliminary commitment phase and enables signers to start the signing process directly with nonces.
- Utilizing a varying number of nonces (1, 2, or 4+) to cater to different security requirements and cryptographic assumptions, such as the algebraic group model (AGM) and the one-more discrete logarithm (OMDL) assumption in the random oracle model (ROM).

The results presented in the paper highlight the efficiency gains and security enhancements achieved through the implementation of MuSig2, making it a valuable contribution to the field of digital signatures and multi-signatures.

On the other hand, the novelty of this research is threefold: First, it integrates EdDSA-inspired deterministic nonce generation into Schnorr multi-signatures, effectively combining EdDSA's robustness with the flexibility and efficiency of Schnorr's signature aggregation. Second, it removes the public nonce (R) from the challenge (c) calculations to ensure the uniqueness of each signature. Third, it allows signers to directly verify possession of secret keys using the aggregated public key, thereby enhancing bandwidth efficiency by eliminating the need for NIZK. This streamlined approach significantly improves security and reduces computational complexity, particularly for devices with constrained resources.

The primary motivation for this research is to address security vulnerabilities inherent in Schnorr multi-signatures, specifically those arising from random nonce generation, such as key cancellation, rogue keys, and nonce reuse attacks. Moreover, this study aims to enhance the efficiency and practical applicability of Schnorr signatures in scenarios where computational resources, memory, and bandwidth are limited.

The research methodology employed mathematical analysis of digital signature algorithms (ECDSA, EdDSA, Schnorr, and MuSig (multi signature) and well-known attacks (such as nonce reuse attacks, virtual machine rewinding attacks, key cancellation attacks, and rogue-key attacks) to understand and assess the effectiveness of the proposed method in addressing various challenges and ascertaining its efficiency. By analyzing the strengths and weaknesses of these digital signature schemes and exploring new techniques, this study presents a novel approach that offers a level of security comparable to that of EdDSA. Consequently, this research contributes to ongoing efforts to

enhance the effectiveness and security of digital signatures.

This study met its objectives by addressing critical issues related to random nonce generation, generating the nonce in a deterministic manner, reducing computational complexity, and decreasing the amount of transmitted data combined with the NIZK proof to better utilize bandwidth.

The paper is organized as follows: Section 2 delves into the fundamentals of digital signatures, covering key algorithms such as the elliptic curve digital signature algorithm (ECDSA), the Edwards-curve digital signature algorithm (EdDSA), and the Schnorr digital signature algorithm. Section 2.4 discusses signature aggregation. Section 3 describes the proposed contributions of this research, and Section 4 concludes the findings presented in this study.

2. Digital signature basics

2.1. Elliptic curve digital signature algorithm ECDSA

Understanding the proposed method in this study requires a thorough explanation of the ECDSA mechanism, as it is a fundamental cornerstone of our approach.

Signature generation and verification algorithms in ECDSA are as follows [6]:

Suppose a sender intends to send a message with a digital signature to a receiver. Initially, they must agree on the curve parameters (E, G, n) and the hashing function H, where E represents the elliptic curve equation used, G represents the elliptic curve base point (the generator point), a point on the curve that generates a subgroup of large prime order n, and n represents the order of G. This means that $nG = O$, where O is the identity element.

In addition to the shared parameters, the process of signature generation involves the following variables: d_A, Q_A, and m. Here, d_A represents the private key, which is randomly selected from the interval [1, n-1]); Q_A represents the public key, computed as $d_A G$; and m represents the message to be signed and sent. Note that the letter 'A' in the variables d_A and Q_A signifies the sender, whom we assume to be Alice.

For Alice to sign a message (m), she follows these steps [6]:

- Calculate e = H(m);
- Select a nonce integer "k" randomly from the range [1, n-1];
- Calculate the point $(x_1, y_1) = kG$;
- Calculate $r \equiv x_1 \pmod n$;
- Calculate $s = k^{-1}(e + rd_A)$;
- The signature is the pair (r, s). Thus, Alice will send the messages(m), Q_A, r and s to Bob.

Note that it is not only required for "k" to be secret, but it is also crucial to select a different "k" for each signature.

Bob follows these steps to verify Alice's signature:

- Calculate e=H(m);
- Calculate $u_1 \equiv es^{-1} \pmod n$ and $u_2 \equiv rs^{-1} \pmod n$;
- Calculate the point $(x_1, y_1) = u_1 G + u_2 Q_A$;
- Alice's signature is valid just if $(x_1, y_1) \equiv kG$.

Proof of the correctness of the algorithm:

$(x_1, y_1) = u_1 G + u_2 Q_A$, since $u_1 \equiv es^{-1} \pmod n$ and $u_2 \equiv rs^{-1} \pmod n$, then:

$$(x_1, y_1) = es^{-1}G + rs^{-1}Q_A = s^{-1}(eG + rQ_A) = \frac{eG + r Q_A}{s},$$

since $s = k^{-1}(e + rd_A)$, then: $(x_1, y_1) = \frac{eG+rQ_A}{K^{-1}(e+rd_A)} = \frac{k(eG+rQ_A)}{e+rd_A}$,

since $Q_A = d_AG$, then: $(x_1, y_1) = \frac{k(eG+rd_AG)}{e+rd_A} = \frac{kG(e+rd_A)}{(e+rd_A)} = kG$.

Note that the ECDSA is susceptible to the nonce reuse attack. This means that if Alice uses the same nonce when signing different messages, an attacker, or the opponent (whom we will refer to as Oscar), can potentially discover Alice's private key (d_A). If Alice reuses the same nonce for different message signings, Oscar can carry out the following steps to expose Alice's private key (d_A):

Alice signs the message (m_1):
(1) Calculate $e_1=H(m_1)$;
(2) Select a nonce "k";
(3) Calculate $(x_1, y_1) = kG$;
(4) Calculate $r \equiv x_1 (\text{mod } n)$;
(5) Calculate $s_1 = k^{-1}(e_1 + rd_A)$;
(6) (m_1) signature is (r, s_1).

Alice signing the message (m_2):
(1) Calculate $e_2=H(m_2)$;
(2) "k" will be the same as assumed.
(3) $(x_1, y_1) = (x_2, y_2) = kG$;
(4) "r" will be the same, $r \equiv x_1 (\text{mod } n)$;
(5) Calculate $s_2 = k^{-1}(e_2 + rd_A)$;
(6) (m_2) signature is (r, s_2).

If Oscar, who was listening to this conversation, can perform a subtraction operation between s_1 and s_2, the resulting outcome will reveal Alice's private key (d_A) in the following manner:

$s_1 - s_2 = k^{-1}(e_1 + rd_A) - (k^{-1}(e_2 + rd_A)) = k^{-1}e_1 + k^{-1}rd_A - k^{-1}e_2 - k^{-1}rd_A = k^{-1}e_1 - k^{-1}e_2$,

$s_1 - s_2 = k^{-1}(e_1 - e_2) = \frac{(e_1 - e_2)}{k}$, $k = \frac{e_1-e_2}{s_1-s_2} = (e_1 - e_2)(s_1 - s_2)^{-1}$.

By knowing the nonce "k", Oscar now knows (s_1, s_2, $e_1=H(m_1)$ and $e_2=H(m_2)$). By substituting them into the equations for (s_1) or (s_2), Oscar can obtain Alice's private key (d_A) as follows:

$s_1 = k^{-1}(e_1 + rd_A)$, $s_1 = \frac{(e_1+rd_A)}{k}$, $ks_1 = e_1 + rd_A$, $rd_A = ks_1 - e_1$, $d_A = \frac{ks_1-e_1}{r}$,

$d_A = (ks_1 - e_1)\, r^{-1}$.

A nonce reuse attack, caused by a random number generator failure, was used to extract the signing key for the PlayStation 3 gaming console [14]. Additionally, a failure in random number generation caused users of the Android Bitcoin Wallet to lose their funds in August 2013 [15]. These incidents emphasize the importance of robust and secure random number generation in cryptographic systems. Thorough testing, evaluation, and implementation of reliable random number generators are crucial to preventing such failures and associated risks.

2.2. Edwards-curve digital signature algorithm (EdDSA)

Understanding the proposed method in this study requires a thorough explanation of the EdDSA mechanism, as it is one of the cornerstone fundamentals of our approach. The signature generation and verification algorithms in EdDSA are as follows [5]:

For Alice to sign a message (m), she follows these steps:

- Select a Private Key (secrete key "S_k") and calculate the public key $P_k=S_kG$;
- Calculate $h = H(S_k)$;
- Calculate the nonce (r) by concatenating (h) with the message (m) and calculate the hash value for them, $r = H(h \parallel m)$, where $\parallel$ means concatenating h with m;
- Calculate R = x-coordinate of (rG);
- Calculate $s = r + H(R\|P_k\|m)S_k$;
- The signature is the pair (R, s), so Alice will send the message (m), P_k, R and s to Bob.

Bob follows these steps to verify Alice's signature:

- Calculate $\tilde{s} = H(R\|P_k\| m)$;
- Calculate $V_1 = sG$, and $V_2 = R + P_k\tilde{s}$;
- The signature is valid just if $V_1 = V_2$.

Proof of the correctness of the algorithm:

$V_2 = R + P_k\tilde{s}$, $R = rG$ and $\tilde{s} = H(R\|P_k\|m)$, then:

$V_2 = rG + P_kH(R \|P_k\| m)$, since $P_k = S_kG$, then:

$V_2 = rG + S_kG\, H(R \|P_k\| m) = G(r + S_kH(R \|P_k\| m))$,

since $s = r + H(R \|P_k\| m)S_k$, then: $V_2 = sG = V_1$.

As observed, EdDSA chooses the nonce deterministically as the hash of a part of the private key and the message. Thus, once a private key is generated, EdDSA has no further need for a random number generator to make signatures. In other words, EdDSA is resistant to nonce reuse attacks if an appropriate hashing function is chosen.

However, it is essential to consider the implications of signing the same message multiple times, leading to the reuse of the same nonce [7]. Repeating the signing process with the same message multiple times utilizes the same nonce repeatedly. This occurs because neither the private key nor the message changes, and hashing them multiple times results in the same hash value.

Nonetheless, this repetition does not compromise the security of the system. The reason lies in the fact that the generated signatures remain identical. Consequently, if Oscar attempts to compare these signatures, the result will be zero, indicating that they are indistinguishable.

For a more comprehensive understanding, let us examine the steps that Oscar might follow to implement a nonce reuse attack:

Alice signs the message (m_1):

(1) Select "S_k" and calculate $P_k=S_kG$;

(2) Calculate $h_1 = H(S_k)$;

(3) Calculate the nonce $r_1 = H(h\|m_1)$;

(4) Calculate R_1 = x-coordinate of (r_1G);

(5) Calculate $s_1 = r_1 + H(R\|P_k\|m_1)S_k$.

(6) (m_1) signature (R_1, s_1).

Alice signs the message (m_1) again:

(1) "S_k" and P_k are the same;

(2) $h_2=h_1 = H(S_k)$;

(3) $r_2=r_1 = H(h \|m_1)$;

(4) $R_2=R_1 = (r_1G) = (r_2G)$;

(5) $s_2=s_1 = r_2 + H(R_2\|P_k\|m_1)S_k$.

(6) (m_1) signature $(R_2,s_2) = (R_1,s_1)$.

Since the two signatures are identical, when Oscar subtracts (s_1) from (s_2), the result would be zero (s_1-s_2= 0). Consequently, Oscar does not receive any benefit or information from these duplicate signatures.

2.3. Schnorr digital signature algorithm

Understanding the proposed method in this study requires a thorough explanation of the Schnorr Digital Signature Algorithm, as it is one of the cornerstone fundamentals of our approach. The interesting aspect of Schnorr signatures lies in their linearity, which offers several advantageous properties. Schnorr signatures follow a specific structure (s=r+ck), where (s) represents the signature, (r) represents the random number, (c) represents the challenge, and (k) represents the secret key. This construction also exhibits linearity, thereby aligning harmoniously with the linear nature of elliptic curve mathematics. The linearity exhibited by Schnorr signatures adds to their appeal and suitability for signature aggregation.

The signature generation and verification algorithms in Schnorr are as follows [16]:

For Alice to sign a message (m), she follows these steps:

- Given a cyclic group G of prime order p with generator g. select a private key (x) such that $0< x < p$;
- Calculate the public key (X), such that $X = g^x$;
- Choose a private nonce (r) randomly, such that $0 < r < p$;
- Calculate public nonce $R = g^r$;
- Calculate c =H (X, R, m);
- Calculate $s \equiv (r + cx) \bmod p$;
- The signature is the pair (R, s). So, Alice will send the message (m), Public Key (X), R and s to Bob.

Bob can calculate (c) since he knows Alice's public key (X), (R), and the message (m). Bob considers Alice's signature valid if $g^s = RX^c$.

Proof of the correctness of the algorithm:

$g^s = RX^c$, since $R = g^r$ and $X= g^x$, then:

$g^s = g^r (g^x)^c = g^{r+xc} = g^s$, because $s = r + xc$.

Note that Schnorr is susceptible to the nonce reuse attack. Oscar can carry out the following steps to expose Alice's private key (x) if she reuses the same nonce for different message signings:

Alice signs the message (m_1):
(1) Select a Private Key (x);
(2) Calculate the Public Key $X= g^x$;
(3) Choose a nonce (r) randomly;
(4) Calculate $R = g^r$;
(5) Calculate $c_1=H (X, R, m_1)$;
(6) Calculate $s_1=r +c_1x$.

Alice signs the message (m_2):
(1) The Private Key (x) is the same;
(2) The Public Key $X = g^x$ is the same;
(3) The nonce (r) is the same as we assumed;
(4) The $R = g^r$ is the same;
(5) Calculate $c_2=H (X, R, m_2)$;
(6) Calculate $s_2=r +c_2x$.

If Oscar performs the subtraction operation between s_1 and s_2, the resulting outcome will reveal Alice's private key (x) in the following manner:

$$s_1 - s_2 = r + c_1x - (r + c_2x) = r + c_1x - r - c_2x = c_1x - c_2x,$$

$$s_1 - s_2 = x (c_1 - c_2),$$

$x = \frac{s_1 - s_2}{c_1 - c_2} = (s_1 - s_2)\,(c_1 - c_2)^{-1}$.

By knowing all the parameters (s_1, s_2, c_1=H (X, R, m_1) and c_2=H (X, R, m_2)), Oscar can obtain Alice's private key (*x*).

2.4. Signature aggregation

To comprehend the reason behind the existence of three rounds in multi-signature schemes and how the proposed methodology in this research can eliminate the second round by demonstrating the knowledge of private keys, it is necessary to explain the basic principles related to signature aggregation, its inherent weaknesses, and the potential for exploitation. First of all, it is essential to familiarize oneself with some fundamental concepts.

The process for creating signatures consistently follows this recipe:

- Generate a secret nonce (r);
- Create a public nonce R, where R = rG;
- Alice sends the message (m), R, and the public key (P=private key(k)G) to Bob.

The actual signature is created by hashing the combination of all the public information above to create a challenge, c = H(R||P||m). The hashing function is chosen so that (e) has the same range as private keys.

The actual signature is created by hashing the combination of all the public information mentioned above to create a challenge, c = H(R||P||m). The hashing function is chosen so that (e) has the same range as the private keys. Now, the signature is constructed using the private information (s = r + ck).

Bob can calculate (c), since he already knows (m), (R), and (P). However, he doesn't know the private key or the nonce.

Given the equation sG = (r+ck) G, this expression is expanded to sG = rG + ckG. It is further simplified to sG=rG + (kG)c. With R =rG and P= kG denoted, the equation simplifies to sG =R + Pc.

So, Bob needs to calculate (sG) and check if this calculated value matches the right-hand side of the equation (R + Pc). Bob already has all the necessary information to perform these steps.

After a brief explanation of the approach used to create signatures and how the receiver can verify their authenticity, let us now discuss signature aggregation:

To gain a simple understanding of this, let's explore the application of the linear property of Schnorr signatures to create a two-of-two multi-signature. In this scenario, Alice and Bob want to co-sign a message without completely trusting each other. They need a way to prove ownership of their individual keys, and the combined signature is valid only if both Alice and Bob contribute to it.

Assuming private keys are denoted k_i and public keys P_i. If we ask Alice and Bob to each supply a nonce, we can try:

$$P_{agg} = P_a + P_b, \tag{1}$$

$$c = H(R_a||R_b||P_a||P_b||m), \tag{2}$$

$$s_{agg} = r_a + r_b + (k_a + k_b)\,c, \tag{3}$$

$$s_{agg} = (r_a + k_a c) + (r_b + k_b c) = s_a + s_b. \tag{4}$$

It appears that both Alice and Bob have the ability to contribute their own (R) values. Additionally, it is possible for anyone to create a two-of-two signature by combining these (R) values with the corresponding public keys. However, it is crucial to note that this scheme suffers from a security vulnerability known as the Key Cancellation Attack.

To understand the key cancellation attack, let us reconsider the previous scenario from a different perspective. This time, Bob has prior knowledge of Alice's public key and public nonce. He waits until she reveals them, then proceeds to deceive by claiming his public key is $P'_b = P_b - P_a$ and his public nonce is $R'_b = R_b - R_a$. It is important to note that Bob does not possess the private keys corresponding to these bogus values.

Despite this deception, everyone assumes that the aggregated signature expression remains $s_{agg} = R_a + R'_b + c(P_a + P'_b)$ in accordance with the aggregation scheme. However, Bob has the ability to create this signature by himself.

To illustrate, we will utilize the equation ($sG = R + Pc$) to explain the vulnerability:

$$s_{agg}G = R_a + R'_b + c(P_a + P'_b) = R_a + (R_b - R_a) + c(P_a + P_b - P_a) = R_b + cP_b = r_bG + ck_bG,$$

$$s_{agg}G = G(r_b + ck_b),$$

$$s_{agg} = (r_b + ck_b), \text{ Therefore, } s_{agg} = s_b.$$

Bob could be compelled to authenticate by signing a message, proving his knowledge of the private keys. This method works, but it requires an additional round of messaging between parties. To prevent this attack, the multi-signature method involves three rounds.

Design of the Schnorr multi-signature scheme:

The primary approach for designing a Schnorr multi-signature scheme can be described as follows:

Consider a scenario involving a group of (n) signers who intend to co-sign a given message (m). Initially, each co-signer randomly calculates and shares a unique value $R_i = g^{r_i}$, where (g) is the generator of the underlying group and r_i is a randomly chosen exponent associated with each co-signer.

Subsequently, each cosigner proceeds to compute two values: $R = \prod_{i=1}^{n} R_i$ and $c = H(\widetilde{X}, R, m)$. Here, $\widetilde{X}$ represents the product of the individual public keys, X_i, of all the cosigners, $\widetilde{X} = \prod_{i=1}^{n} X_i$ (aggregated public key).

Moving on, the partial signature for each cosigner is computed as $s_i = r_i + cx_i$, where x_i represents the secret key associated with cosigner i. Each cosigner independently calculates their respective partial signature.

Finally, all partial signatures are merged into a single signature using the relation $s \equiv \sum_{i=1}^{n} s_i \pmod p$, where (p) is the group's prime modulus.

The verification process of a signature, denoted as (R, s), on a message (m), with respect to a set of public keys $\{X_1, ..., X_n\}$, can be expressed equivalently as the equation $g^s = R\widetilde{X}$.

It should be noted that this verification equation corresponds precisely to the verification equation utilized in Schnorr signatures with respect to the public key represented by $\widetilde{X}$. This particular property is commonly referred to as key aggregation. However, it is essential to acknowledge that these protocols are susceptible to a rogue-key attack [17]. This attack occurs when a malicious signer deliberately sets their public key to $X_1 = g^{x_1}(\prod_{i=2}^{n} X_i)^{-1}$, thereby enabling the signer to generate signatures for the individual public keys $\{X_1, ..., X_n\}$ independently.

MuSig signature scheme:

MuSig is a Schnorr-based multi-signature scheme that allows a group of signers to produce a short, joint signature on a common message. MuSig is provably secure in the plain public-key model.

The signature generation and verification algorithms in MuSig are as follows [18]:

MuSig is parameterized by group parameters (G, p, g) and three hash functions (H_{com}, H_{agg} and H_{sig}). These three hash functions can be either identical or distinct, with the decision depending on the desired security level. Utilizing distinct hash functions enhances the level of security.

- Round 1:

A group of (n) signers wants to co-sign a message (m). Let X_1 and x_1 be the public and private keys of a specific signer, where $X_1 = g^{x_1}$. Let $X_2, \ldots, X_n$ be the public keys of the other co-signers and let (L) be the multi-set of all public keys involved in the signing process.

For $i \in \{1, \ldots, n\}$, the signer computes $a_i = H_{agg}(L, X_i)$ as well as the "aggregated" public key $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i}$.

- Round 2:

The signer generates a random private nonce r_1, computes the public nonce $R_1 = g^{r_1}$, and the commitment $t_1 = H_{com}(R_1)$, then sends t_1 to all other cosigners.

Upon receiving the commitments $t_2, \ldots, t_n$ from other cosigners, the signer sends R_1 to all other cosigners. This ensures that the public nonce is not exposed until all commitments have been received.

Upon receiving $R_2, \ldots, R_n$ from other cosigners, the signer verifies that $t_i = H_{com}(R_i)$ for all $i \in \{2, \ldots, n\}$.

By sending the commitments first, the signers can ensure that no one can change their public nonce after seeing the others. This prevents a rogue-key attack, as a malicious signer would need to know the public nonces of the other signers in advance to construct a rogue public key. If any signer detects a mismatch, they abort the protocol. This ensures that the signers are using the same set of public nonces to compute the aggregated signature, which is important for the security and correctness of MuSig.

- Round 3:

If all commitment and random challenge pairs can be verified with H_{agg}, the next step involves computing $R = \prod_{i=1}^{n} R_i$, $c = H_{sig}(\tilde{X}, R, m)$ and $s_1 \equiv (r_1 + ca_1x_1) \bmod p$.

Signature s_1 is sent to all other cosigners. Upon receiving $s_2, \ldots, s_n$ from other cosigners, the signer can compute $s \equiv \sum_{i=1}^{n} s_i \pmod p$. The signature is the pair (R, s).

In order to verify the aggregated signature, the message (m), multi-set of public keys (L), (R), and (s) will be sent to the receiver.

The receiver follows these steps to verify the aggregated signature:

- $a_i = H_{agg}(L, X_i)$ for $i \in \{1, \ldots, n\}$;
- $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i}$;
- $c = H_{sig}(\tilde{X}, R, m)$;
- The signature is accepted if $g^s = R \prod_{i=1}^{n} X_i^{ca_i} = R\tilde{X}^c$.

Proof of the correctness of the algorithm:

$g^s = R\,\tilde{X}^c$, since $R = \prod_{i=1}^{n} R_i = \prod_{i=1}^{n} g^{r_i}$, and since $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i} = \prod_{i=1}^{n} (g^{x_i})^{a_i}$, then:

$g^s = \prod_{i=1}^{n} g^{r_i} \prod_{i=1}^{n} (g^{x_i a_i})^c = \prod_{i=1}^{n} g^{r_i + x_i a_i c} = g^{r_1 + x_1 a_1 c} + \ldots + g^{r_n + x_n a_n c} = g^{\sum_{i=1}^{n} s_i} = g^s$, because $s = \sum_{i=1}^{n} s_i$.

This method is immune to key cancellation attacks and rogue key attacks, but it is vulnerable to the nonce reuse attack. To understand how MuSig is susceptible to this attack, consider the scenario

where Alice, among a group of cosigners, reuses the same nonce for signing different messages. In this case, Oscar can carry out the following steps to expose Alice's private key (x):

Alice signing the message (m_1):
(1) Select a Private Key (x_1);
(2) Calculate the Public Key $X_1 = g^{x_1}$;
> Round_1:
(3) Calculate $a_1 = H_{agg}(L, X_1)$;
(4) $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i}$;
> Round_2:
(5) Choose a nonce (r_1) randomly;
(6) Calculate $R_1 = g^{r_1}$;
(7) Calculate $t_1 = H_{com}(R_1)$;
(8) Send t_1 to other cosigners and receive $t_2, \ldots, t_n$;
(9) Send R_1 to other cosigners and receive $R_2, \ldots, R_n$, then calculate and check t_i;
> Round_3:
(10) Calculate:

- $R = \prod_{i=1}^{n} R_i$;
- $c = H_{sig}(\tilde{X}, R, m_1)$;
- $s_1 = r_1 + ca_1x_1$
- Send s_1 to other cosigners and receive s_i from them to calculate:

$$s \equiv \sum_{i=1}^{n} s_i$$

Alice signing the message (m_2):
(1) The Private Key (x_1) is the same;
(2) The Public Key X_1 is the same;
> Round_1:
(3) Calculate $a_1' = a_1 = H_{agg}(L, X_1)$;
(4) $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i}$;
> Round_2:
(5) The nonce (r_1) is the same as assumed;
(6) $R_1' = R_1$;
(7) Calculate $t_1' = t_1 = H_{com}(R_1)$;
(8) Send t_1' to other cosigners and receive$t_2, \ldots, t_n$;
(9) Send R_1' to other cosigners and receive $R_2, \ldots, R_n$, then calculate and check t_i;
> Round_3:
(10) Calculate:

- $R = \prod_{i=1}^{n} R_i$;
- $c' = H_{sig}(\tilde{X}, R, m_2)$;
- $s_1' = r_1 + c'a_1x_1$
- Send s_1' to other cosigners and receive s_i from them to calculate:

$$s \equiv \sum_{i=1}^{n} s_i$$

If Oscar listens to this conversation, he can perform the subtraction operation between these two signings (s_1) and (s_1'), and the resulting outcome will reveal Alice's private key (x_1) in the following manner:

$$s_1 - s_1' = r_1 + ca_1x_1 - (r_1 + c'a_1x_1) = r_1 + ca_1x_1 - r_1 - c'a_1x_1,$$

$$s_1 - s_1' = ca_1x_1 - c'a_1x_1 = a_1x_1 (c - c'),$$

$$x_1 = (s_1 - s_1')(a_1(c - c'))^{-1}.$$

Since Oscar knows (s_1, s_1', R, m_1 and m_2) and can calculate $c = H_{sig}(\tilde{X}, R, m_1)$, $c' = H_{sig}(\tilde{X}, R, m_2)$ and $a_1 = H_{agg}(L, X_1)$, he can then calculate Alice's private key (x_1).

To gain a different perspective on the problem, we can examine it from an alternative angle. It is essential to consider the following scenario where Alice and Bob try to co-sign a message (m_1):

Alice, Signing the message (m_1):
(1) Select a Private Key (x_1);
(2) Calculate the Public Key $X_1 = g^{x_1}$;
(3) Choose the private nonce (r_1) randomly;
(4) Calculate the public nonce $R_1 = g^{r_1}$;
(5) Send R_1 to Bob and receive R_2;
(6) Calculate $R = R_1 + R_2$;
(7) Calculate $c = H_{sig}(X, R, m_1)$;
(8) Calculate $s_1 = r_1 + ca_1x_1$.
(9) Send s_1 to Bob.
(10) For any reason, the signing did not complete.
(11) m_1 signing process has been resumed.
(12) Send R_1 to Bob again;
(13) Receive $R_2^{'}$ from Bob;
(14) Computes:

- $\tilde{R} = R_1 + R_2^{'}$;
- $c^{'} = H_{sig}(\tilde{X}, \tilde{R}, m)$;
- $s_1^{'} = r_1 + c^{'}a_1x_1$

(15) Send $s_1^{'}$ to Bob.
… etc.

Bob, Signing the message (m_1):
(1) Select a Private Key (x_2);
(2) Calculate the Public Key $X_2 = g^{x_2}$;
(3) Choose the private nonce (r_2) randomly;
(4) Calculate the public nonce $R_2 = g^{r_2}$;
(5) Send R_2 to Bob and receive R_1;
(6) Calculate $R = R_1 + R_2$;
(7) Calculate $c = H_{sig}(X, R, m_1)$;
(8) Calculate $s_2 = r_2 + ca_2x_2$;
(9) For any reason, the signing did not complete.
(10) m_1 signing process has been resumed:
- Choose new nonce ($r_2^{'}$) randomly;
- Calculate $R_2^{'} = g^{r_2^{'}}$ and send to Alice

… etc.

If Oscar listens to this conversation, then he can perform the subtraction operation between Alice's two signatures. The resulting outcome will reveal Alice's private key (x_1) in the following manner:

$$s_1 - s_1^{'} = r_1 + ca_1x_1 - r_1 - c^{'}a_1x_1 = ca_1x_1 - c^{'}a_1x_1 = x_1 a_1 (c - c^{'}),$$

$$x_1 = \frac{s_1 - s_1^{'}}{a_1(c - c^{'})} = (s_1 - s_1^{'})(a_1(c - c^{'}))^{-1}.$$

This highlights the vulnerability of the MuSig protocol to nonce reuse attacks when the same nonce is utilized in multiple signature operations.

Based on the previous explanation, the novelty in the MuSig signature scheme involves the addition of the component a_i to address the issue of key cancellation attacks in key aggregation for multi-signature schemes. Additionally, the MuSig signature scheme adopts a three-round signing process to counter rogue key attacks, which represent the main drawback of this method. As a result, alternative models for multi-signature schemes that require only two rounds, such as MuSig-DN and MuSig2, have been proposed.

As mentioned in the Introduction section [12], the complexity of the NIZK proof in the MuSig-DN protocol renders it impractical for use on dedicated signing devices like hardware wallets, which are typically employed for storing bitcoins.

Additionally, according to the Introduction section of this study [13], MuSig2 can use different numbers of nonces depending on the specific configuration:

- MuSig2 can use one nonce, but this configuration may have limitations in terms of security guarantees.
- MuSig2 can use two nonces to be secure under the Algebraic One-More Discrete Logarithm (AOMDL) assumption in the ROM when AGM is additionally assumed.
- MuSig2 can use four or more nonces to be proven secure under the (OMDL) assumption in the ROM (Random Oracle Model).

Thus, each signer in MuSig2 provides a list of at least two nonces, which are then combined to form the aggregate nonce for the signing operation.

In the ROM, each cosigner_i utilizes four or more nonces $(R_i^{`}, R_i^{``}, R_i^{```}, R_i^{````}, \ldots, R_n^n)$,. Then, the cosigner_i uses a random combination $R_i = R_i^{`} (R_i^{``})^b (R_i^{```})^{b^2} (R_i^{````})^{b^3}$, where the exponent *b* is set by hashing essentially the entire protocol input and transcript after the nonce exchange round (the aggregated public key, the message, and the nonces of all signers):

$$b = H(\tilde{X}, m, (\prod_{i=1}^{n} R_i^{`}, \prod_{i=1}^{n} R_i^{``}, \prod_{i=1}^{n} R_i^{```}, \prod_{i=1}^{n} R_i^{````}, \ldots, \prod_{i=1}^{n} R_i^n)). \quad (5)$$

In the AGM, each cosigner_i utilizes two nonces $(R_i^{`}, R_i^{``})$. Then, the $\text{cosigner}_i$ uses a random combination $R_i = R_i^{`} (R_i^{``})^b$, where $b = H(\tilde{X}, m, (R_1^{`}R_2^{`}, R_1^{``}R_2^{``}))$.

When using two or four nonces in the MuSig2 scheme, as compared to using a single nonce, there is some additional computational complexity involved. This is because each signer, providing two or four nonces in MuSig2, faces an overhead in combining these nonces to form the aggregate nonce. This process involves additional computations to derive the scalar *b* via a hash function. Additionally, the signer's complexity increases slightly due to the need to handle multiple nonces and perform the necessary calculations for nonce aggregation.

3. Proposed contributions

This section elucidates the proposed solution for addressing the nonce generation problem existing in Schnorr's Digital Signature and Schnorr's Multi Signature schemes, as explained in Sections 2.3 and 2.4. The proposed solution includes:

- Utilizing a similar private key generation approach as seen in the digital signature algorithm (EdDSA) to generate deterministic private nonces. This study suggests generating the private nonce by combining the message with the hashed value of the private key (*x*).
- Excluding the public nonce (R) from the calculation of the challenge $c = H_{sig}(X, R, m)$. To ensure the uniqueness of each signature, additional context and protocol-level safeguards akin to those utilized in the EdDSA framework (which falls beyond the purview of this study) should be implemented. This includes the integration of sequence numbers, timestamps, and other session-specific data into the messages. These components, which operate externally to the digital signature mechanism, are intended to mitigate certain attacks, notably replay attacks.
- In the proposed multi-signature scheme, the aggregated public key $\tilde{X}$ has been utilized instead of a NIZK proof to demonstrate that the owner of the public keys knows the corresponding secret keys. Therefore, if the aggregated public key $\tilde{X}$ matches any of the public keys of one or more cosigners, then the signing process is rejected.

The process for implementing these solutions is described in two sections:

First: Schnorr Digital Signature algorithm:

The signature generation and verification process in Schnorr is as follows:

To sign a message (m), Alice follows these steps:

- Select a Private Key (x) such that $0 < x < p$;
- Calculate the Public Key (X), such that $X = g^x$;
- Calculate $h=H(x)$;
- Calculate the private nonce $r = H(h||m)$;
- Calculate the public nonce $R = g^r$;
- Calculate $c = H(X, m)$;
- Calculate $s \equiv (r + cx) \bmod p$;
- The signature is the pair (R, s). Thus, Alice will send the message (m), public key (X), (R), and (s) to Bob.

Bob considers Alice's signature valid only if $gs = RX^c$.

Proof of the correctness of the algorithm:

$g^s = RX^c$, since $R = g^r$ and $X= g^x$, then $g^s = g^r (g^x)^c = g^{r+xc} = g^s$, where $s = r + xc$.

In the current scenario, Alice is unable to sign two different messages using the same nonce. This is because the private nonce changes when the message changes. However, if Alice signs the same message twice, it implies that she is using the same nonce. This raises a concern about the potential compromise of system security. For a more comprehensive understanding, let us examine the steps that Oscar might follow to implement a nonce reuse attack:

Alice signs the message (m_1):

(1) Select a Private Key (x_1);

(2) Calculate the Public Key $X_1= g^x$;

(3) Calculate $h = H(x_1)$;

(4) Calculate private nonce $r_1=H(h||m_1)$;

(5) Calculate the public nonce $R_1= g^{r_1}$;

(6) Calculate $c_1=H(X_1, m_1)$;

(7) Calculate $s_1= r_1+c_1x_1$;

(8) The signature is the pair (R_1, s_1).

Alice signs the message (m_1) again:

(1) The Private Key (x_1) is the same;

(2) The Public Key X_1 is the same;

(3) $h = H(x_1)$ is the same as we assumed;

(4) private nonce $r_2=r_1= H(h||m_1)$;

(5) Calculate the public nonce $R_2=R_1$;

(6) Calculate $c_2=c_1=H(X_1, m_1)$;

(7) Calculate $s_1 = s_2= r_2+c_2x_2$;

(8) The signature is the pair (R_1, s_1).

This shows that the two signatures, s_1 and s_2, are identical, and subtracting one from the other results in zero (s_1-s_2=0). Therefore, there is no benefit or information gained from duplicate signatures. This means that the security of the system is not compromised when the same message is signed twice with the same nonce. The proposed method effectively prevents nonce reuse attacks by deterministically calculating the private nonce, ensuring it changes with each new message. Even if the same message is signed multiple times, the process remains secure.

In addition, the proposed method for signing messages does not require storing any information during the signing process. The necessary calculations can be made from the inputs and received messages, eliminating the risk of manipulation through virtual machine rewinding attacks.

To prevent potential exploitation of collision probabilities, it is recommended to use appropriate cryptographic hashing functions.

Second: Proposed multi-signature scheme:

In order to differentiate multi-signature schemes, the classification is as follows:

- Multi-signature schemes, wherein n-of-n signers are required to collectively produce a valid signature.
- Threshold signature schemes wherein any subset of size t-of-n signers out of n total signers can collectively produce a valid signature.

It should be noted that this section exclusively addresses multi-signature schemes.

The proposed multi-signature generation and verification scheme is as follows:

- Round 1:

A group of n signers want to co-sign a message m. Let X_1 and x_1 be the public and private key of a specific signer, where $X_1=g^{x_1}$. Let X_2, …, X_n be the public keys of the other co-signers, and let (L) be the hash value of the multi-set of all public keys involved in the signing process, $L=H_{agg}(X_1||...||X_n)$:

- For $i\in\{1, \ldots, n\}$, each signer computes the following: $a_i=H_{agg}(L, X_i)$, as well as the aggregated public key: $\widetilde{X}=\prod_{i=1}^{n} X_i^{a_i}$. The signer must check if ($\widetilde{X} = X_i^{a_i}$); if so, then the signing operation is terminated.
- The signer calculates $h=H_{agg}(x_1)$, computes the private nonce $r_1= H_{agg}(h||m)$, computes the public nonce $R_1=g^{r_1}$, and sends R_1 to all other co-signers and receives R_2, …, R_n from other co-signers.

- Round 2:

- Calculate $c = H_{sig}(\widetilde{X}, m)$ and $s_1 \equiv r_1+ca_1x_1 \pmod p$;
- Signature s_1 is sent to all other co-signers. Upon receiving s_2, …, s_n from other co-signers, they then calculate $R = \prod_{i=1}^{n} R_i$ and $s \equiv \sum_{i=1}^{n} s_i \pmod p$. The signature will be the pair (R, s). The message (m) and signature pair (R, s) will be sent to the receiver.

The receiver follows these steps to verify the aggregated signature:

- $a_i=H_{agg}(L, X_i)$ for $i\in\{1, \ldots, n\}$;
- $\widetilde{X}= \prod_{i=1}^{n} X_i^{a_i}$;
- $c = H_{sig}(\widetilde{X}, m)$;
- The signature is accepted if $g^s = R\,\widetilde{X}^c$.

Proof of the correctness of the algorithm:

$g^s = R\widetilde{X}^c$, since $R= \prod_{i=1}^{n} R_i=\prod_{i=1}^{n} g^{r_i}$, and since $\widetilde{X}= \prod_{i=1}^{n} X_i^{a_i} = \prod_{i=1}^{n}(g^{x_i})^{a_i}$, then:

$g^s = \prod_{i=1}^{n} g^{r_i}\ (\prod_{i=1}^{n} g^{x_i a_i})^c = \prod_{i=1}^{n} g^{r_i+x_i a_i c} = g^{r_1+x_1 1c} + \ldots + g^{r_n+x_n a_n c}$,

$g^s =g^{\sum_{i=1}^{n} s_i}= g^s$, because $s = \sum_{i=1}^{n} s_i$.

To enhance comprehension of the operational mechanism of the proposed method and its resilience against the rogue key attack, a simple example is employed to elucidate it:

Suppose that Alice and Bob want to co-sign a message (m) and then send this multi-signature to Robert (the receiver). The steps for the proposed multi-signature scheme are as follows:

- Round 1:

Let X_a and x_a be the public and private keys of Alice, where $X_a=g^{x_a}$, and let X_b and x_b be the public and private keys of Bob, where $X_b=g^{x_b}$, then $L = H_{agg}(X_a||X_b)$:

- Alice computes $a_a=H_{agg}(L, X_a)$, and because she already knows Bob's public key X_b (which is publicly available), she can calculate $a_b = H(L||X_b)$. After this, Alice calculates the aggregated public key $\widetilde{X}= X_a^{a_a} \cdot X_b^{a_b}$.

If Alice finds that $\widetilde{X}=X_b^{a_b}$, then the protocol is terminated because the public key that Bob uses is fake, equating to $X_b= (g^{x_b})^{a_b} \cdot (X_a^{a_a})^{-1}$. This manipulation suggests Bob's attempt to execute a

rogue key attack, exploiting his knowledge of Alice's public key to undermine the integrity of the aggregated public key, calculated as $\widetilde{X}= X_a^{a_a} \cdot X_b^{a_b}= X_a^{a_a} \cdot g^{x_b} \cdot (X_a^{a_a})^{-1} = g^{x_b} = X_b^{a_b}$. By halting the signing process at this juncture, the protocol effectively prevents Bob from forging the signature by nullifying Alice's public key contribution, thus safeguarding against potential signature forgery by Bob.

Similarly, Bob computes $a_b=H_{agg}(L, X_b)$, and because he already knows Alice's public key X_a(which is publicly available), he can calculate $a_a= H(L||X_a)$. After this, Bob calculates the aggregated public key $\widetilde{X}= X_a^{a_a} \cdot X_b^{a_b}$.

If Bob finds that $\widetilde{X}=X_b^{a_b}$, then the protocol is terminated, indicating that Alice might be attempting a rogue key attack.

- If neither Alice nor Bob are performing a rogue key attack, then the multi-signature steps will proceed. Alice calculates $h=H_{agg}(x_a)$, private nonce $r_a= H_{agg}(h||m)$, Public nonce $R_a=g^{r_a}$, and sends R_a to Bob and receives R_b from him, who calculated R_b in the same way as Alice did.

- Round 2:

- Alice calculates $R = \prod_{i=1}^{n} R_i$, $c = H_{sig}(\widetilde{X}, m)$ and $s_a \equiv (r_a+ca_ax_a) \bmod p$.
- Alice sends the signature s_a to Bob and receiving s_b from him, who calculates s_b in the same manner as Alice did.
- The aggregated signature will be $s = s_a + s_b \pmod p$. Then, the multi-signature will be the pair (R, s). Subsequently, the message (m) and the signature pair (R, s), will be sent to Robert.

Robert will consider the multi-signature to be valid only if $g^s = R\ \widetilde{X}^c$

When evaluating the efficacy of the proposed method against nonce reuse attacks, it should be noted that Alice is unable to sign two different messages using the same nonce. This is because the private nonce changes when the message does. However, if Alice signs the same message twice, it indicates she is using the same nonce. This raises concern about whether this compromises the security of the system.

Let us assess whether the security of the system would be compromised in instances where Alice co-signs a message (m) twice with Bob while utilizing the same nonce:

Alice signs the message (m):
(1) Select a Private Key (x_a);
(2) Calculate the Public Key $X_a=g^{x_a}$
> Round_1;
(3) Calculate $L = H_{agg}(X_a||X_b)$,
Calculate $a_a= H_{agg}(L, X_a)$;
(4) $\widetilde{X}= \prod_{i=1}^{n} X_i^{a_i} = X_a^{a_a} \cdot X_b^{a_b}$,
Check if $\widetilde{X}=X_b^{a_b}$;
(5) Calculates h=H (x);
(6) Calculate $r_a= H(h||m)$;

Alice signs the same message (m) again:
(1) The Private Key (x_a) is the same;
(2) The Public Key X_a is the same
> Round_1;
(3) Calculate $L = H_{agg}(X_a||X_b)$,
Calculate $a_a^{\prime}= a_a= H_{agg}(L, X_a)$;
(4) $\widetilde{X}= \prod_{i=1}^{n} X_i^{a_i}$ is the same;
(5) $h^{\prime}= h = H(x)$;
(6) $r_a^{\prime}= r_a= H(h||m)$;
(7) $R_a^{\prime} =R_a= g^{r_a}$, is the same;

(7) Calculate $R_a = g^{r_a}$;
(8) Send R_a to Bob and receive R_b
> Round_2;
(9) Calculate $R = \prod_{i=1}^{n} R_i = R_a \cdot R_b$,

$c = H_{sig}(\tilde{X},m)$, and $s_a = r_a + ca_a x_a$,

Send s_a to Bob and receive s_b from him, and the aggregated signature will be:
$s \equiv \sum_{i=1}^{n} s_i \pmod p = s_a + s_b$.

(8) send $R_a^{|}$ to Bob and receive R_b;
> Round_2
(9) Calculate $R = \prod_{i=1}^{n} R_i = R_a^{|} \cdot R_b$

$= R_a \cdot R_b$, $c = H_{sig}(\tilde{X},m)$ and

$s_a^{|} = s_a = r_a + ca_a x_a$,
Send s_a to Bob and receive s_b from him, and the aggregated signature will be:
$s \equiv \sum_{i=1}^{n} s_i \pmod p = s_a + s_b$.

Since the two signatures ($s_a^{|}$ and s_a) are identical, this means that when Oscar subtracts one signature (s_a) from the other ($s_a^{|}$), the result is zero (s_a-$s_a^{|}$=0). Consequently, this indicates that the system's security remains uncompromised when signing the same message with the same nonce multiple times, as the duplicate signatures yield no new information.

Now, the effectiveness of the proposed method in addressing the issue explained in the second scenario in Section 2.4 will be assessed as follows:

Alice, signing the message (m):
(1) Select a Private Key (x_a);
(2) Calculate the Public Key $X_a = g^{x_a}$;
> Round_1;
(3) Calculate $L = H_{agg}(X_a||X_b)$
Calculate $a_a = H_{agg}(L, X_a)$;

(4) Calculate $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i} = X_a^{a_a} \cdot X_b^{a_b}$

Check if $\tilde{X}=X_b^{a_b}$;

(5) Calculates $h = H_{agg}(x_a)$;
(6) Calculate $r_a = H_{agg}(h||m)$;
(7) Calculate $R_a = g^{r_a}$;
> Round_2;
(8) Send R_a to Bob and receive R_b;
(9) Calculate:

- $R = \prod_{i=1}^{n} R_i = R_a \cdot R_b$
- $c = H_{sig}(\tilde{X},m)$
- $s_a = r_a + ca_a x_a$

(10) Send s_a to Bob and receive s_b.
(11) For any reason, the signing did not complete.

Bob, signing the message (m):
(1) Select a Private Key (x_b);
(2) Calculate the Public Key $X_b = g^{x_b}$;
> Round_1;
(3) Calculate $L = H_{agg}(X_a||X_b)$
Calculate $a_b = H_{agg}(L, X_b)$;

(4) Calculate $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i} = X_a^{a_a} \cdot X_b^{a_b}$

Check if $\tilde{X}=X_b^{a_b}$;

(5) Calculates $h = H_{agg}(x_b)$;
(6) Calculate $r_b = H_{agg}(h||m)$;
(7) Calculate the public nonce $R_b = g^{r_b}$;
> Round_2;
(8) Send R_b to Alice and receive R_a;
(9) Calculate $R = \prod_{i=1}^{n} R_i = R_a \cdot R_b$,

$c = H_{sig}(\tilde{X},m)$ and $s_b = r_b + ca_b x_b$;

(10) For any reason, the signing did not complete.
(11) m signing process has been resumed, Calculate:

(12) m signing process has been resumed.
(13) Send same R_a to Bob again and receive R_b' from Bob;
(14) Computes:

- $R' = \prod_{i=1}^{n} R_i = R_a \cdot R_b'$,
- $c' = H_{sig}(\tilde{X}, m) = c$,
- $s_a' = r_a + c' a_a x_a = s_a$.

(15) Send s_a' to Bob and receive
… etc.

- $a_b = H_{agg}(L, X_b)$,
- $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i}$, $h = H_{agg}(x_b)$,
- $r_b' = H_{agg}(h \| m) = r_b$,
- $R_b' = g^{r_b'} = R_b$ and send to it

Alice and receive R_a from him
… etc.

In the suggested approach, Bob cannot choose different nonces for the same message. Even if Bob attempts to do so intentionally, Alice will still sign the same message with the same unique nonce, resulting in identical signatures. Specifically, the signatures (s_a' and s_a) are identical because we excluded the public nonce from the challenge calculations. Consequently, subtracting one signature from the other results in zero ($s_a - s_a' = 0$). This means that Oscar gains no benefit or information from these duplicate signatures.

Thus, the proposed method effectively thwarts nonce reuse attacks by deterministically calculating and changing the private nonce for each new message. Additionally, the method does not require storing any information during the signing process, as all necessary calculations can be derived from the inputs and received messages. This eliminates the risk of manipulation through virtual machine rewinding attacks. To mitigate the potential exploitation of collision probabilities, employing suitable cryptographic hashing functions is recommended.

4. Conclusions

The method proposed to enhance the Schnorr digital signature and two-round MuSig shows promise in achieving several important objectives. First, it improves the Schnorr digital signature by combining the private key and the message through hashing, then it hashes the result again to generate a private nonce, inspired by the method used in EdDSA.

The proposed method excludes the public nonce (R) from the challenge calculations ($c = H_{sig}(X, R, m)$), ensuring the uniqueness of each signature. Moreover, removing (R) from these calculations reduces the computational complexity, a crucial factor in constrained resources devices where every computational step matters. A significant advantage of this method is that it ensures the private nonce will be different for each new message. Even if the same message is signed multiple times with the same nonce, it will not make the system insecure, thus effectively solving the issue of random number generation failures.

In multi-signature schemes, the proposed method compares to the MuSig signature scheme. While MuSig is resistant to key cancellation and rogue key attacks, it involves three rounds and is susceptible to nonce reuse attacks. In contrast, the proposed method is a two-round scheme that is

immune to key cancellation, rogue key, and nonce reuse attacks, offering enhanced security with lower computational complexity.

When compared to MuSig-DN, it is clear that MuSig-DN introduces inherent complexity due to the necessity of deploying NIZK proofs to validate the deterministic generation of nonces. This requirement not only increases computational overhead but also adds complexity to the implementation, making the protocol less practical for devices with constrained resources. In contrast, the proposed method utilizes the aggregated public key $\widetilde{X}$ to demonstrate ownership of the corresponding secret keys. This approach eliminates the need to compute and send NIZK proofs along with signatures, leading to reduced computational complexity and bandwidth usage. Thus, the proposed method is more efficient in terms of computational complexity and bandwidth utilization, especially for devices with constrained resources.

Comparing the proposed method to MuSig2, the key difference lies in how they handle nonces. MuSig2 allows for the use of multiple nonces based on a configured set where each signer needs to provide at least two nonces. These nonces are then combined to create an aggregated nonce for the signing process. Thus, generating and using multiple nonces in MuSig2 adds computational complexity, particularly in deriving the scalar "b" using a hash function. Also, sending multiple nonces in MuSig2 leads to higher bandwidth usage. In contrast, the proposed method employs only a single nonce. This streamlined approach eliminates additional computational tasks and minimizes bandwidth requirements, making the process more efficient and straightforward compared to MuSig2.

Although this study does not definitively assert superiority over the established and proven MuSig-DN and MuSig2 methods, it represents a valuable contribution to ongoing efforts to improve digital signature technology. In the proposed method, the public nonce (R) serves the receiver's verification process to ensure the signature's validity. Additionally, the proposed method removes the need to store any information between signing rounds, meaning signers don't have to remember details such as the nonce or partial signature. They can generate these values again using public inputs and messages. This benefit reduces storage and computational requirements for signers and enhances resistance against virtual machine rewinding attacks. As a result, this approach prevents cheating in the public keys, reduces computational complexity, and minimizes the amount of information sent combined with NIZK proof, leading to better utilization of bandwidth.

Author contributions

Nawras H. Sabbry: Conceptualization, methodology, formal analysis, investigation, writing (original draft preparation and visualization); Alla Levina: Supervision, writing (review and editing), project administration, funding acquisition. All authors have read and agreed to the published version of the manuscript.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This research was funded by the Ministry of Science and Higher Education of the Russian Science Foundation (Project "Goszaadanie" No.075-00003-24-02, FSEE-2024-0003).

Conflict of interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

References

1. W. Diffie, M. Hellman, New directions in cryptography, *IEEE Trans. Inf. Theory*, **22** (1976), 644–654. https://doi.org/10.1109/TIT.1976.1055638
2. R. L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Commun. ACM*, **21** (1978), 120–126. https://doi.org/10.1145/359340.359342
3. F. Pub, Digital signature standard (DSS), 1994. Available from: https://csrc.nist.gov/pubs/fips/186/upd1/final
4. C. P. Schnorr, Efficient signature generation by smart cards. *J. Cryptol*., **4** (1991), 161–174. https://doi.org/10.1007/BF00196725
5. D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, B. Y. Yang, High-speed high-security signatures, *J. Cryptogr. Eng.*, **2** (2012), 77–89. https://doi.org/10.1007/s13389-012-0027-1
6. D. B. Johnson, A. Menezes, S. A. Vanstone, The elliptic curve digital signature algorithm (ECDSA), *Int. J. Inf. Secur*., **1** (2001), 36–63.
7. S. Josefsson, I. Liusvaara, Edwards-curve digital signature algorithm (EdDSA), *IRTF. RFC*, 2017, 8032. https://doi.org/10.17487/RFC8032. ISSN 2070-1721
8. M. Beunardeau, A. Connolly, H. Ferradi, R. Géraud-Stewart, D. Naccache, D. Vergnaud, Reusing nonces in Schnorr signatures, *Pro. Cryptology-AFRI*., 2017, 224–241. https://doi.org/10.1007/978-3-319-66402-6_14
9. Y. Romailler, S. Pelissier, Practical fault attack against the Ed25519 and EdDSA signature schemes, *Proc. Workshop Fault Diag. Tole. Cryp.*, 2017, 17–24. https://doi.org/10.1109/FDTC.2017.12
10. K. Chalkias, F. Garillot, Y. Kondi, V. Nikolaenko, Non-interactive half-aggregation of EdDSA and variants of Schnorr signatures, *Lecture Notes Comp. Sci.*, 2021, 12704. https://doi.org/10.1007/978-3-030-75539-3_24
11. Y. Kondi, C. Orlandi, L. Roy, Two-round stateless deterministic two-party Schnorr signatures from pseudorandom correlation functions, *Lecture Notes Comp. Sci.*, 2023, 14081. https://doi.org/10.1007/978-3-031-38557-5_21
12. J. Nick, T. Ruffing, Y. Seurin, P. Wuille, MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces, *Conf. Comput. Commun. Security*, 2020, 1717–1731. https://doi.org/10.1145/3372297.3417236
13. J. Nick, T. Ruffing, Y. Seurin, MuSig2: Simple two-round Schnorr multi-signatures, *Lecture Notes Comp. Sci.*, 2021, 12825. https://doi.org/10.1007/978-3-030-84242-0_8
14. P. Q. Nguyen, I. E. Shparlinski, The insecurity of the elliptic curve digital signature algorithm with partially known nonces, *Des. Codes Cryptogr.*, **30** (2003), 201–217. https://doi.org/10.1023/A:1025436905711
15. Online content: Android security vulnerability, 2013. Available from: https://bitcoin.org/en/alert/2013-08-11-android
16. D. Boneh, Schnorr digital signature scheme, *Lecture Notes Comp. Sci.*, 2005, 541–542. https://doi.org/10.1007/0-387-23483-7_369

17. M. Michels, P. Horster, On the risk of disruption in several multiparty signature schemes, *Lecture Notes Comp. Sci.*, 1996.
18. G. Maxwell, A. Poelstra, Y. Seurin, P. Wuille, Simple Schnorr multi-signatures with applications to Bitcoin, *Des. Codes Cryptogr.*, 2019. https://doi.org/10.1007/s10623-019-00608-x

AIMS Press