Multi-Signature Scheme Resistant to Randomness Injection Attacks - A Bitcoin Case

Łukasz Krzywiecki
Department of Fundamentals of Computer Science
Wrocław University of Science and Technology
Wrocław, Poland
lukasz.krzywiecki@pwr.edu.pl

Adam Połubek
Department of Fundamentals of Computer Science
Wrocław University of Science and Technology
Wrocław, Poland
222278@student.pwr.edu.pl

Hannes Salin
Department of Information and Communication Technology
Swedish Transport Administration
Borlänge, Sweden
hannes.salin@trafikverket.se

2021 IEEE 20th International Symposium on Network Computing and Applications (NCA) | 978-1-6654-9550-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/NCA53618.2021.9685253

***Abstract*—We propose a modification of a multi-signature scheme, which was previously used as an enhancement of multi-signatures for the Bitcoin cryptocurrency. Our scheme is secure in a new stronger security model in which we allow the forger to inject or control the ephemeral (randomness) values in the end-user's signing device. Thus, our modified scheme is resistant to ephemeral key leakage attacks. We also provide a proof of concept implementation of our scheme, providing a time complexity and performance analysis.**

***Index Terms*—bitcoin, Schnorr signatures, multi-signature schemes, key aggregation, ephemeral secret setting, ephemeral secret leakage.**

I. Introduction

In the context of Bitcoin a *multi-signature* is referred to a distributed *m-of-n transaction*, i.e. m signers, out of a maximum n, are needed to sign the transaction. In order to perform transactions within Bitcoin, a stack-based scripting system called *Script* is used. It uses a set of commands (or *opcodes*) for instructing sequential transaction executions. Requiring multiple signers for a transaction, i.e. multi-signatures, is indicated by the opcode `OP_CHECKMULTISIG` with a corresponding `OP_CHECKMULTISIGVERIFY` flag. These m-of-n procedures output a binary response, indicating whether the transaction was valid or not. Given m signatures, the size of the resulting joint output grows linearly over m. To improve computational and space complexity, a Schnorr-based scheme MuSig was proposed in [1]. It introduced a *key aggregation*: the joint signature can be verified as a standard Schnorr signature over a single *aggregated public key*, computed from all m individual public keys. The key aggregation technique improves the privacy of all involved cosigners, since they cannot be linked via the aggregated public key.

A. Problem Statement and Threat Description

In Schnorr-based schemes the resulting outputs are linear combinations of random values, challenges, and the user's private keys. Such schemes are secure, only under the assumption that the hardware (and the underlying operation system) is trusted, and the randomness, used during the scheme execution, is not leaked. Computations are usually performed in a less secure area, where the ephemeral values can be accessed. A malicious hardware- or software implementer can access or tamper the ephemerals via hidden channels, perform alteration, or even set these values in advance. This would allow for secret key extraction and subsequent forgery in a worst case scenario. We consider any type of device that can handle internal secure storage, i.e an architecture separating the user code area from the secure area - typically a *Hardware Security Module* (HSM) as depicted in Fig. 1. When the device needs to compute a signature, the code queries the HSM for the user's secret key x. One possible attack scenario is where the

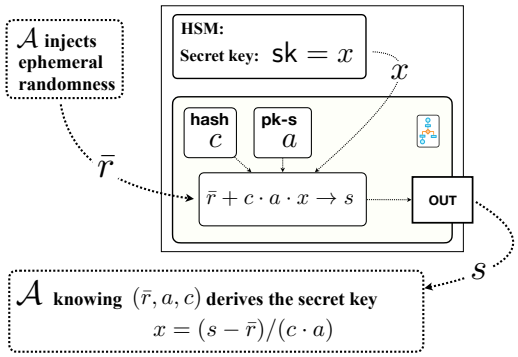

Fig. 1. *Randomness setup* attack on MuSig from [1].

adversary manage to inject or learn the ephemeral randomness $\bar{r}$ in the device, during the signature phase. In practice, it is not unlikely to have devices with poor pseduo-random generators producing low entropy, thus generating predicatble bits which the adversary can learn. Even if the HSM securely stores the secret key sk, the knowledge of the ephemeral $\bar{r}$ allows to *unmask* and derive sk from the resulting signature, e.g. as shown in Fig. 1. Obviously, having sk, the adversary is now able to impersonate the device.

978-1-6654-9550-9/21/$31.00 ©2021 IEEE

B. Contribution

The security of the MuSig scheme from [1], based on the Schnorr signature scheme [2], relies on the hardness of the discrete logarithm problem (DLP). The scheme is vulnerable to ephemeral secret leakages, as a session randomness masks the secret key in the resulting signature. We propose a modification of the MuSig construction which mitigates that problem. For applications, which require interactivity with a designated trusted party, it is possible to transform a single user construction into its multi-signature version, using the methodology proposed in this paper. Note that the above-mentioned constructions, are randomness based with the explicit interactive commitment phase.

C. Related Work

Bit leakage during computations was first addressed by Chari et al. [3]. The analysis continued independently by Goubin and Patarin [4] and later by Alwen et al. [5]. The problem of bit leakage but from cryptographic keys was analyzed by Canetti et al. [6]. Ephemeral key leakage has been considered for several types of identification and key exchange protocols [7]–[10]. Multi-signatures for blockchain were considered in [1], [11], [12], and recently in [13], MuSig-DN [14] and VT-Schnorr [15]. Unfortunately, these schemes are still vulnerable to ephemeral key leakage as presented in our stronger security model.

II. System Settings and Security Requirements

A. Protocol Flow

The proposed scheme run as a partial interactive protocol between signers. In the preparatory phase each signer computes an aggregated public key to be used for the signature creation. Then each signer computes a commitment to a random value in the exponent (i.e. g^{r_1} for the participant with index 1). Commitments are broadcast to all cosigners. Only after all commitments are received, their inputs are broadcast. Thus all cosigners can verify that all random values in the exponent are precomputed individually and independently by all participants (steps 1-4 of Sign in Fig. 3). In the second phase of the signature creation, the signer computes a partial signature S_1 which is broadcast. Upon reception of the cosigner's partial signatures, a final aggregated multi-signature σ is computed, representing steps 5-7 in Fig. 3.

B. Hardware Security Modules and Minimal Functionality

We consider HSMs, which not only stores the secret key, but also perform a specific mathematical operation with it, defined as a *minimal function* $\mathbf{f}$. Hence it is referred to as a secure computation inside the HSMs. Distinct devices, indexed with $\{1,\ldots,n\}$, using their own secrets $\mathsf{sk}_1,\ldots,\mathsf{sk}_i,\ldots,\mathsf{sk}_n$, run corresponding functions $\mathbf{f}_1,\ldots,\mathbf{f}_i,\ldots,\mathbf{f}_n$ within their own HSMs. In order to address the ephemeral key leakage problem, we consider a strengthened security protocol where the HSM minimal function handles exponentiation with the secret key sk. Minimal function $\mathbf{f}(x)$ takes an input x from a less secure area of the device, and returns x^{sk} for further computations. Note that in this case sk is hidden in the exponent, and any attacker face the hardness of DLP to extract sk from the result. As described in Fig. 2 the attacker is still able to inject an ephemeral value of its choice. However, with the fresh generator $\hat{g}$, the minimal function $\mathbf{f}$ computes $\hat{g}^x$, served to the less secure area where the signature $\hat{S} = \hat{g}^{\bar{r}}(\mathbf{f}(\hat{g}))^{ca} = \hat{g}^{\bar{r}+cax}$ is created. Therefore the attacker obtains a linear equation in the exponent for a generator $\hat{g}$ which is different in each signing session.

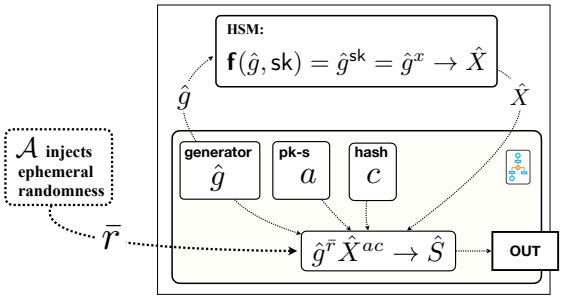

Fig. 2. Device architecture signing procedure.

III. Multi-Signatures with Stronger Security

Let $x_i \leftarrow_{\$} X$ denote that x_i is sampled from a uniformly distributed set X. Let $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{A}$ be a hash function modeled in the *Random Oracle Model* (ROM), where the output are elements in the set $\mathbb{A}$. Let $\mathcal{G}(1^\lambda)$ be an algorithm that takes a security parameter 1^λ as input and outputs a tuple $G = (q, g, \mathbb{G})$, where q is a prime number, and $\langle g\rangle = \mathbb{G}$ is a multiplicative group of order q.

Definition 1 (Bilinear Pairing)**.** *Let $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ be groups with generators g_1, g_2, g_T respectively. Let $q = |\langle g_1\rangle| = |\langle g_2\rangle| = |\langle g_T\rangle|$. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be a pairing function with the following properties:*

1) *Bilinearity: $\forall(a, b \in \mathbb{Z}_q^*, g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2)$: $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$,*
2) *Computability: Computing $\hat{e}$ is efficient,*
3) *Non-degeneracy: $\exists(g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2) : \hat{e}(g_1, g_2) \neq 1$.*

If $\mathbb{G}_1 = \mathbb{G}_2$ we say that the pairing is symmetric.

The security of our scheme is based on: Discrete Logarithm Problem (DLP), and Gap Computational Diffie-Hellman (GDH), which we omit due to space constraints.

Definition 2 (Multi-signature Scheme)**.** *A multi-signature scheme* MSS *is a system which consists of four algorithms:* ParGen, KeyGen, $\pi(\{\mathsf{Sign}((\mathsf{sk}_i, \mathsf{pk}_i), L, m)\}_1^n)$ *and* Verify.

$\mathsf{params} \leftarrow \mathsf{ParGen}(1^\lambda)$**:** *takes a security parameter λ as input, and outputs public parameters* params *available to all users of the system.*

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$**:** *outputs the secret key* sk *and corresponding public key* pk.

$\sigma \leftarrow \pi(\{\mathsf{Sign}((\mathsf{sk}_i, \mathsf{pk}_i), L, m)\}_1^n)$**:** *denotes the signature protocol run by the ith participant, which requires an*

MuSig scheme	Modified MuSig scheme
$\mathsf{ParGen}(1^\lambda)$: $\mathbb{G} = (q, g, G) \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{params} = (q, g, G)$.	$\mathsf{ParGen}(1^\lambda)$: Let $\mathbb{G} = (q, g, G, G_T, \hat{e}) \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{params} = (q, g, G, G_T, \hat{e})$.
$\mathsf{KeyGen}()$: Generate random $\mathsf{sk} = x \leftarrow \mathbb{Z}_q^*$, compute $\mathsf{pk} = X = g^x$ and output $(\mathsf{sk}, \mathsf{pk})$.	
$\pi(\{\mathsf{Sign}((x_1, X_1), L = \{X_1, \ldots, X_n\}, m)\}_1^n)$: 1. Calculate $a_i = \mathcal{H}_{agg}(L, X_i)$ for each $i \in \{1, \ldots, n\}$. 2. Compute aggregated public key $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i}$. 3. Generate random $r_1 \leftarrow_\$ \mathbb{Z}_q^*$, compute $R_1 = g^{r_1}$, $t_1 = \mathcal{H}_{com}(R_1)$, and send commitment t_1 to others. 4. Await for $t_2, \ldots, t_n$ from other cosigners, then send R_1. Upon reception of $R_2, \ldots, R_n$, check if for each $i \in \{2, \ldots, n\}$ equation $t_i = \mathcal{H}_{com}(R_i)$ holds. Abort if for any it is false, otherwise continue.	
5. Compute: $R = \prod_{i=1}^{n} R_i$, $c = \mathcal{H}_{sig}(\tilde{X}, R, m)$, $s_1 = r_1 + ca_1x_1$, and send s_1 to other cosigners. 6. Wait until reception of $s_2, \ldots, s_n$, then compute $s = \sum_{i=1}^{n} s_i$. 7. Output signature $\sigma = (R, s)$.	5. Compute: $R = \prod_{i=1}^{n} R_i$, $c = \mathcal{H}_{sig}(\tilde{X}, R, m)$, $\hat{g} = \mathcal{H}_g(\tilde{X}, R, m)$, $S_1 = \hat{g}^{r_1}(\mathbf{f}_1(\hat{g}))^{ca_1} = \hat{g}^{r_1 + ca_1x_1}$, and send S_1 to other cosigners. 6. Wait until reception of $S_2, \ldots, S_n$, then compute $\hat{S} = \prod_{i=1}^{n} S_i$. 7. Output signature $\sigma = (R, \hat{S})$.
$\mathsf{Verify}(L = \{X_1, \ldots, X_n\}, m, \sigma)$: 1. Calculate $a_i = \mathcal{H}_{agg}(L, X_i)$ for each $i \in \{1, \ldots, n\}$. 2. Compute aggregated public key $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i}$.	
3. Get $c = \mathcal{H}_{sig}(\tilde{X}, R, m)$. 4. Output 1 if $g^s = R\tilde{X}^c$, 0 otherwise.	3. Get $c = \mathcal{H}_{sig}(\tilde{X}, R, m)$, $\hat{g} = \mathcal{H}_g(\tilde{X}, R, m)$. 4. Output 1 if $\hat{e}(\hat{S}, g) = \hat{e}(\hat{g}, R\tilde{X}^c)$, 0 otherwise.

Fig. 3. MuSig multi-signature scheme and our modified version.

input of its key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$*, the public keys of all signers* $L = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ *, and message* m*. The final output is the signature* σ*.*

$1 \ / \ 0 \leftarrow \mathsf{Verify}(L, m, \sigma)$**:** *denotes the verification algorithm, where the input is the set of public keys* $L = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$*, message* m*, and signature* σ*. It outputs* 1 *when verification is successful,* 0 *otherwise.*

We recall the MuSig multi-signature scheme from [1] in Fig. 3. The scheme is presented from the perspective of the signer with index 1. Procedures $\mathcal{H}_{agg}$, $\mathcal{H}_{com}$, $\mathcal{H}_{sig}$ are hash functions used for generation of the aggregation exponent a_i, commitment t_i, and signature mask c.

In our stronger security model for MSS we assume that a forger $\mathcal{F}$ can corrupt all but one devices, i.e. we assume w.l.o.g. that it is given all secret keys except sk_1 of the device with index 1.

Definition 3 (Chosen Ephemeral Supervisory Forgery (CESF))**.** *Let* $\mathsf{MSS} = (\mathsf{ParGen}, \mathsf{KeyGen}, \pi(\{\mathsf{Sign}((\mathsf{sk}_i, \mathsf{pk}_i), L, m)\}_1^n), \mathsf{Verify})$ *be a multi-signature scheme. We define the security experiment* $\mathtt{Exp}_{\mathsf{MSS}}^{\mathsf{CESF},\lambda,\ell}$*:*

`Init stage` : *Let* $\mathsf{params} \leftarrow \mathsf{ParGen}(1^\lambda)$, $\{(\mathsf{sk}_i, \mathsf{pk}_i)\}_1^n \leftarrow \mathsf{KeyGen}()$ *and* $\mathcal{F}$ *be a malicious probabilistic polynomial time (PPT) algorithm given the set of public/private key-pairs*

`Hash Oracles` : *Hash oracles are setup to serve output from hash queries during signature creation and verification.*

`Sign Oracle` : $\mathcal{O}_{\mathsf{Sign}}(m, L, \bar{r}_1)$ *is initialized with keys* $(\mathsf{sk}_1, \mathsf{pk}_1)$*, participates in a polynomial number* ℓ *executions of signature protocol using the injected* $\bar{r}_1$*.*

`Forger` : $\mathcal{F}^{\mathcal{O}}$ *is a malicious PPT algorithm given the key-pairs* $\{(\mathsf{sk}_i, \mathsf{pk}_i)\}_2^n$*, i.e. starting from index* 2*, and the public key* pk_1*.* $\mathcal{F}^{\mathcal{O}}$ *does not have the secret* sk_1*. Having access to hash oracles and* $\mathcal{O}_{\mathsf{Sign}}$*, it executes the signature protocol a polynomial number* ℓ *times.* $\mathsf{v}^{\mathbf{r}(\ell)} = \{T_1, \ldots, T_\ell\}$ *is the view that* $\mathcal{F}$ *gains after* ℓ *runs of the signature protocol, where* T_i *is the transcript of ith execution with* $\bar{r}_1$ *injected to uncorrupted device.* $\mathcal{F}^{\mathcal{O}}$ *goal is to produce a signature* σ^* *over message* m^* *not previously queried to* $\mathcal{O}_{\mathsf{Sign}}$*, which is positively verifiable with public keys* L *containing* pk_1*.*

We define the advantage of $\mathcal{F}^{\mathcal{O}}$ *in the experiment* $\mathtt{Exp}_{\mathsf{MSS}}^{\mathsf{CESF},\lambda,\ell}$ *as the probability of acceptance of a forged signature in the* Verify *procedure:*

$$\mathbf{Adv}(\mathcal{F}^{\mathcal{O}}, \mathtt{Exp}_{\mathsf{MSS}}^{\mathsf{CESF},\lambda,\ell}) = \mathtt{Pr}\left[\begin{array}{l} \mathsf{params} \leftarrow \mathsf{ParGen}(1^\lambda), \\ L = \{(\mathsf{sk}_i, \mathsf{pk}_i)\}_1^n \leftarrow \mathsf{KeyGen}(), \\ (m^*, \sigma^*) \leftarrow \mathcal{F}^{\mathcal{O}}\left(\{(\mathsf{sk}_i, \mathsf{pk}_i)\}_2^n, \mathsf{pk}_1, \mathsf{v}^{\mathbf{r}(\ell)}\right) \\ \mathsf{Verify}(L, m^*, \sigma^*) = 1 \end{array}\right].$$

We say that the multi-signature scheme MSS *is secure if* $\mathbf{Adv}(\mathcal{F}^{\mathcal{O}}, \mathtt{Exp}_{\mathsf{MSS}}^{\mathsf{CESF},\lambda,\ell})$ *is negligible in* λ*.*

As shown in Fig. 1, the MuSig scheme from [1], recalled in Fig. 3, is not secure in the proposed CESF model.

IV. Modified MuSig Construction

The main modification is in the Sign procedure (Step 5 of Fig 3). We put s_1 in the exponent of a new generator $\hat{g} = \mathcal{H}_g(\tilde{X}, R, m)$, unique in every signature. Sending $S_1 = \hat{g}^{s_1}$, computed as $\hat{g}^{r_1}(\mathbf{f}_1(\hat{g}))^{ca_1}$, by the means of the HSM with minimal functionality $\mathbf{f}_1$ (which exponentiates the input $\hat{g}$ with the stored secret sk_1), makes it hard to obtain the secret key even if the attacker has acces to randomness r_1. Verification is done using the bilinear map $\hat{e}$ to check the equality $\hat{e}(\hat{S}, g) = \hat{e}(\hat{g}, R\tilde{X}^c)$.

A. Sketch of Security Analysis

The forger $\mathcal{F}^{\mathcal{O}}$ have access to the hash oracles and is enabled to query the sign oracle $\mathcal{O}_{\mathsf{Sign}}(m, L, \bar{r}_1)$ with the power of injecting ephemeral values $\bar{r}_1$, and thus gaining the view $\mathsf{v}^{\mathbf{r}(\ell)}$. Under the assumption that the advantage of the forger is non-negligible, we fork the execution of a forgery

TABLE I
PERFORMANCE AND SCALABILITY ANALYSIS. PROCEDURES WITH STAR ASTERISK($*$) DENOTE OUR MODIFIED SCHEME AND n IS THE NUMBER OF COSIGNERS.

Operation	Time (ms)	Pre	Sig	Ver	Pre*	Sig*	Ver*	Cosigners	Pre*	Sig*	Ver*
G1:mul	0.0001	$n-1$	$n-1$	n	$n-1$	$2n+1$	1	$n=1$	39.275	9.377	8.239
G1:exp	1.7425	$n+1$	-	$n+2$	$n+1$	2	1	$n=10$	46.354	9.427	8.172
G1:hashTo	3.9371	-	-	-	-	1	1	$n=30$	126.935	9.512	8.100
Pairing	1.2449	-	-	-	-	-	2	$n=100$	494.586	10.996	8.588

attempt on the oracle $\mathcal{O}_{\mathcal{H}_{sig}}$ query, returning a different signature hash c'. That value will help us to break the GDH with non-negligible probability.

Theorem 1. *The modified* MuSig *scheme shown in Fig. 3 is secure (in the sense of Definition 3).*

Proof. We omit the proof due to the space constraints. □

V. IMPLEMENTATION

Our implementations uses the Python Charm cryptography library [16] with the SS512 curve. All benchmarks were performed on a MacBook Pro, 2.7 GHz Dual-Core Intel i5. The results are presented in Tab. I, in terms of three fundamental phases: Pre (steps 1-4 in Fig. 3), Sig (steps 5-7 of Sign in Fig. 3), and Ver (steps 1-4 of Verify in in Fig. 3). We also present a scalability analysis of each phase total running time, with the perspective of number of cosigners (n) in Tab. I. All timings were collected as the average running time over 100 iterations, and measured in milliseconds (ms). Note that the performance analysis neglected communication time and network delays between parties. The analysis is performed from the perspective of one signer collaborating with the rest of the group, i.e. a signer with index 1 collaborating with n cosigners.

VI. CONCLUSION

In this paper we modified and improved the MuSig scheme from [1] in such way that it becomes immune to the ephemeral key leakage setting. MuSig scheme was originally designated for Bitcoin, however the blockchain technology is still promising for many other types of applications. Enhancements in multi-signature schemes used with blockchains (such as modifications of the MuSig scheme for Bitcoin) could thus potentially impact other distributed signature-based use cases, like ITS and V2X applications.

VII. ACKNOWLEDGMENT

The research was partially financed from the Fundamental Research Fund nr 8201003902 of the Wrocław University of Science and Technology.

REFERENCES

[1] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple Schnorr multi-signatures with applications to Bitcoin," *Designs, Codes and Cryptography*, vol. 87, pp. 2139–2164, Sep 2019.

[2] C.-P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[3] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Advances in Cryptology — CRYPTO' 99* (M. Wiener, ed.), (Berlin, Heidelberg), pp. 398–412, Springer Berlin Heidelberg, 1999.

[4] L. Goubin and J. Patarin, "DES and Differential Power Analysis The "Duplication" Method," in *Cryptographic Hardware and Embedded Systems* (Ç. K. Koç and C. Paar, eds.), (Berlin, Heidelberg), pp. 158–172, Springer Berlin Heidelberg, 1999.

[5] J. Alwen, Y. Dodis, and D. Wichs, "Leakage-resilient public-key cryptography in the bounded-retrieval model," in *Advances in Cryptology - CRYPTO 2009* (S. Halevi, ed.), (Berlin, Heidelberg), pp. 36–54, Springer Berlin Heidelberg, 2009.

[6] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai, "Exposure-resilient functions and all-or-nothing transforms," in *EUROCRYPT*, 2000.

[7] Ł. Krzywiecki, A. Bobowski, M. Słowik, M. Słowik, and P. Kozieł, "Schnorr-like identification scheme resistant to malicious subliminal setting of ephemeral secret," *Computer Networks*, vol. 179, p. 107346, 2020.

[8] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *Provable Security* (W. Susilo, J. K. Liu, and Y. Mu, eds.), (Berlin, Heidelberg), pp. 1–16, Springer Berlin Heidelberg, 2007.

[9] J. Lee and J. H. Park, "Authenticated key exchange secure under the computational diffie-hellman assumption.," *IACR Cryptol. ePrint Arch.*, vol. 2008, p. 344, 2008.

[10] B. Ustaoglu, "Obtaining a secure and efficient key agreement protocol from (h)mqv and naxos." Cryptology ePrint Archive, Report 2007/123, 2007. https://ia.cr/2007/123.

[11] C. Ma and M. Jiang, "Practical lattice-based multisignature schemes for blockchains," *IEEE Access*, vol. PP, pp. 1–1, 12 2019.

[12] Y. Zhao, "Practical aggregate signature from general elliptic curves, and applications to blockchain," *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019.

[13] J. Nick, T. Ruffing, and Y. Seurin, "MuSig2: Simple Two-Round Schnorr Multi-signatures," in *Advances in Cryptology – CRYPTO 2021* (T. Malkin and C. Peikert, eds.), (Cham), pp. 189–221, Springer International Publishing, 2021.

[14] J. Nick, T. Ruffing, Y. Seurin, and P. Wuille, "Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, (New York, NY, USA), p. 1717?1731, Association for Computing Machinery, 2020.

[15] S. A. K. Thyagarajan, A. Bhat, G. Malavolta, N. Döttling, A. Kate, and D. Schröder, "Verifiable timed signatures made practical," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, (New York, NY, USA), p. 1733?1750, Association for Computing Machinery, 2020.

[16] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.