Extremely Simple
Fail-Stop ECDSA Signatures

Mario Yaksetig

BitFashioned
mario@bitfashioned.com

Abstract. Fail-stop signatures are digital signatures that allow a signer to prove that a specific forged signature is indeed a forgery. After such a proof is published, the system can be stopped.
We introduce a new simple ECDSA fail-stop signature scheme. Our proposal is based on the minimal assumption that an adversary with a quantum computer is not able to break the (second) preimage resistance of a cryptographically-secure hash function. Our scheme is as efficient as traditional ECDSA, does not limit the number of signatures that a signer can produce, and relies on minimal security assumptions. Using our construction, the signer has minimal computational overhead in the signature producing phase and produces a signature indistinguishable from a 'regular' ECDSA signature.

Keywords: Fail-stop signatures · Formal Methods Analysis · ECDSA.

1 Introduction

Presently, the Internet relies heavily on the Elliptic Curve Digital Signature Algorithm (ECDSA) as this signature scheme represents the backbone for the issuance of digital signatures online. This issue is even more relevant in the cryptocurrency landscape, where the majority of blockchain platforms use ECDSA to secure their transactions. Effectively, this signature scheme secures a hundreds of billions of dollars since the cryptocurrency funds of each user are secure only as long as no one can steal their funds. Therefore, strengthening the ECDSA scheme, which secures these funds, is of extreme importance.

In this work, we introduce a new protocol that allows legitimate users to prove whether or not a specific signature is a forgery. This is particularly useful in a setting involving a dispute between the real user and an adversary who maliciously obtained the secret key.

1.1 Motivation

Blockchains are not designed to be easily upgraded, and such upgrades are traditionally very controversial. To cause no friction and no need for any type of hard forks, we introduce a new fail-stop variant of the ECDSA scheme that allows users to prove whether or not a signature using a specific key is legitimate or not without requiring any change to existing ECDSA verification algorithms. This is useful as it implies that smart contracts and existing wallet and node infrastructure can remain unchanged.

Fail-stop signatures, which can detect and halt forgery attempts, represent an innovative step forward in cryptographic security. In a world where ECDSA-based cryptocurrencies face relentless security breaches leading to substantial financial losses, the integration of fail-stop signatures could provide a powerful defense. This approach significantly reduces the risk of fraudulent transactions, and enables a dispute resolution in case of a key compromise.

1.2 Our Contributions

We introduce a very lightweight addition to the traditional ECDSA to encode quantum-secure secret key information in the nonce used to produce each ECDSA signature. This encoding allows the signer to, in an event of a dispute, selectively open the nonce used in that signature and prove whether or not a signature is well-formed. Unlike the original failstop signatures [4], our proposal does not set a limit on signatures that can be constructed using the same secret key and does not affect the signature size, which remains exactly the same as a traditional ECDSA signature. Our approach can be integrated in any elliptic-curve based digital signature scheme and also complements existing fallback designs as proposed by Chaum et al. [2, 1].

Arguably, the most important part of our construction is that an adversary able to break the ECDLP is not able to obtain the secret preimage used to generate the nonce for any individual signature. This feature allows the real owner of the signing keys to prove that a specific signature is a forgery.

1.3 Fail-stop Signatures

Fail-stop signatures operate similarly to traditional digital signatures. The signer has a secret key that is used to produce signatures. These signatures can then be verified by any party who knows the corresponding public key. A signature that is successfully verified under the public key is considered acceptable. Fail-stop signatures introduce an additional property where a signer can prove that a specific signature is forged. Therefore, in the case of a dispute, a signer can provide a judge with a proof of forgery. The judge can then test if the signature is correct and provide a verdict of whether or not the provided signature is a forgery. If the proof is accepted, then there is substantial evidence that the computational assumption of the system is broken.

Definition 1 (Fail-stop Signatures). *A fail-stop signature scheme (FSS) is a 5-tuple* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Test}, \mathsf{Prove}, \mathsf{Verify})$*, where:*

- $\mathsf{Gen}(\mathsf{params}, \mathsf{r_A}, \mathsf{r_C}) \rightarrow (\mathsf{sk}, \mathsf{pk})$. *This is a polynomial-time two-party protocol for generating the keys. The protocol is executed by the signer A, and a trusted center C, who both get params* $= (k, \lambda, N)$ *as input. Furthermore,* *each party has a secret random string, r_A and r_C. N is the maximal number of signatures that the signer is willing to construct using the same secret key. k is the security parameter for the recipient and λ is the security parameter for the signer.*

- $\mathsf{Sign(sk, i, m_i)} \rightarrow \sigma$. *This is a polynomial-time algorithm that on input the secret key sk, a message number $i \leq N$, and a message sequence $m = (m_1, ..., m_i)$ from $\mathcal{M}$, constructs a signature on m_i if the previously signed messages were $\{m_1, ..., m_{i-1}\}$. The output σ is called the correct signature.*
- $\mathsf{Test(pk, m, \sigma)} \rightarrow$ OK/NotOK. *This is a polynomial-time algorithm that on input the public key pk, a message $m \in \mathcal{M}$, and a signature σ on m outputs either* OK *or* NotOK. *If* $\mathsf{Test(pk, m, \sigma)} =$ OK*, then σ is an acceptable signature on m.*
- $\mathsf{Prove(sk, m, \sigma', hist)} \rightarrow \pi$. *This is a polynomial-time algorithm that on input the secret key sk, a message $m \in \mathcal{M}$, a possible signature σ' on m, and the history hist of previously signed messages (plus their signatures) either outputs the string $\pi =$ "not a forgery" or a bit string proof $\pi \in \{0, 1\}^*$.*
- $\mathsf{Verify(pk, m, \sigma', \pi)} \rightarrow$ Accept/Reject. *This is a polynomial-time algorithm that on input the public key, a message $m \in \mathcal{M}$, a possible signature σ' on m and a string proof π outputs either* Accept *or* Reject. *If the result is* Accept*, the proof π is called a valid proof of forgery.*

A proof of forgery is always non-interactive so that it can subsequently be shown to others, and the system can be stopped in consensus. The proof must satisfy two requirements[1]:

- The ability to prove forgeries must work independently of the computational power of potential forgers.
- It must be infeasible for the signer to construct signatures that she can later prove to be forgeries[2].

Since it is equally important that fail-stop signatures cannot be forged, the signer still has to take part in choosing the keys. However, the recipients of signatures must be sure that the signer cannot disavow her own signatures. It is therefore necessary that the recipients or a center trusted by the recipients also participate in the key generation.

2 Simple Fail-stop ECDSA

The core idea behind our construction is that a signer can secretly hide special information when generating the nonce for each ECDSA signature.

Our design proposes two simple changes to the traditional ECDSA protocol. First, the signer generates an additional secret value α. Second, the signer uses

[1] It can be shown that these two properties imply security against forgery

[2] We skip this requirement as we do not desire the involvement of additional parties during key generation. Our construction, however, also supports this functionality.

this extra secret value and hashes it along with the message and its index, to generate the nonce for each signature. This ensures that the nonce is unique for every message and that it is random, assuming the hash function behaves as a random oracle. We refer the reader to Table 1 and Table 2 for a detailed protocol description.

Keygen(1^λ)	Sign(α, i, m)	Test($\mathtt{pk}, \mathsf{m}, \sigma$)
$\alpha \xleftarrow{\$} \mathbb{Z}_q$	$z \leftarrow H(\mathsf{m})$	Parse: $(r, s) \xleftarrow{p} \sigma$
$\mathtt{sk} \leftarrow H(\alpha)$	$k \leftarrow H(\alpha, i, m)$	If $(r, s) \notin \mathbb{Z}_q$
$\mathtt{pk} \leftarrow \mathtt{sk} \cdot G$	$(e_x, e_y) \leftarrow k \cdot G$	Return NotOK
return $(\alpha, \mathtt{pk})$	$r \leftarrow e_x \mod p$	$w \leftarrow s^{-1}$
	If $r = 0 \mod p$	$z \leftarrow H(\mathsf{m})$
	Pick another k	$u_1 \leftarrow zw \mod p$
	and start again	$u_2 \leftarrow rw \mod p$
	$s \leftarrow k^{-1} \cdot (z + r \cdot \mathtt{sk})$	$(e_x, e_y) \leftarrow u_1 \times G + u_2 \times \mathtt{pk}$
	If $s = 0 \mod p$	If $(e_x, e_y) = (0, 0)$
	Pick another k	Return NotOK
	and start again	If $r = e_x \mod p$
	Return $\sigma = (r, s)$	Return OK

Table 1. Extended ECDSA Construction

3 Discussion

Verifpal Formal Verification. We analysed our construction using Verifpal [3]. We defined an active attacker, where the adversary is in charge of delivering the messages and define a quantum adversary. To do so, we use the *leak* command and expose all the secret values that are considered 'protected' by the ECDLP. The tool output that regardless of the compromise of the ECDSA secret key value, only the honest owner of the secret preimage α is able to produce well-formed signatures.

Implementation. We are implementing the protocol as an extension to EthSigner. Upon completion, the team intends to obtain a security audit by independent reputable external parties.

Formal Security Proofs. Presently, we are producing the adequate complete proofs of security of our construction. The next step is to, upon completing the formalization, publish an extended version exposing the construction and corresponding proofs.

$\mathsf{Prove}(\alpha, m, \sigma', hist)$	$\mathsf{Verify}(\mathtt{pk}, \mathsf{m}, \sigma, \pi)$
Parse: $(r, s) \overset{p}{\leftarrow} \sigma'$ Parse: $i \overset{p}{\leftarrow} hist'$ $k \leftarrow H(\alpha, i, m)$ $(e_x, e_y) \leftarrow k \cdot G$ $r' \leftarrow e_x \mod p$ If $r' \neq r$ $\quad \pi \leftarrow (\alpha, i)$ If $r' = r$ $\quad \pi \leftarrow$ "Not a forgery" Return π	Parse: $\pi' \overset{p}{\leftarrow} \pi$ If π' = "Not a forgery" $\quad$ Return Reject If $\pi' = (\alpha, i)$ $\quad \mathtt{sk}' \leftarrow H(\alpha)$ $\quad \mathtt{pk}' \leftarrow sk' \cdot G$ $\quad$ If $\mathtt{pk}' \neq pk$ $\quad\quad$ Return Reject $\quad$ If $\mathtt{pk}' = pk$ $\quad\quad$ Parse: $(r, s) \overset{p}{\leftarrow} \sigma$ $\quad\quad k \leftarrow H(\alpha, i, m)$ $\quad\quad (e_x, e_y) \leftarrow k \cdot G$ $\quad\quad r' \leftarrow e_x \mod p$ $\quad\quad$ If $r' \neq r$ $\quad\quad\quad$ Return Reject

Table 2. Prove and Verify algorithms of Fail-Stop ECDSA

4 Conclusion

We introduce a new very simple and cheap fail-stop signature scheme. Our construction is very useful and practical, however, slightly differs from the original fail-stop signature definition for usability and practical deployment reasons. We expect this work to set a foundation for a fruitful debate around key compromise in the cryptocurrency space.

References

1. Chaum, D., Larangeira, M., Yaksetig, M.: Tweakable sleeve: A novel sleeve construction based on tweakable hash functions. Cryptology ePrint Archive, Paper 2022/888 (2022), https://eprint.iacr.org/2022/888
2. Chaum, D., Larangeira, M., Yaksetig, M., Carter, W.: W-ots(+) up my sleeve! a hidden secure fallback for cryptocurrency wallets. Cryptology ePrint Archive, Paper 2021/872 (2021), https://eprint.iacr.org/2021/872
3. Kobeissi, N., Nicolas, G., Tiwari, M.: Verifpal: Cryptographic protocol analysis for the real world. In: Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop. p. 159. CCSW'20, Association for Computing Machinery, New York, NY, USA (2020)
4. Pedersen, T.P., Pfitzmann, B.: Fail-stop signatures. SIAM J. Comput. **26**(2), 291–330 (apr 1997). https://doi.org/10.1137/S009753979324557X, https://doi.org/10.1137/S009753979324557X