Hybrid Post-Quantum Signatures in Hardware Security Keys

Diana Ghinea[1,2], Fabian Kaczmarczyck[2], Jennifer Pullman[2], Julien Cretin[2], Stefan Kölbl[2], Rafael Misoczki[2], Jean-Michel Picod[2], Luca Invernizzi[2], Elie Bursztein[2]

[1] ETH ghinead@ethz.ch

[2] Google {dianamin,kaczmarczyck,jpullman,cretin,kste,rafaelmisoczki,jmichel,invernizzi,elieb}@google.com

Abstract. Recent advances in quantum computing are increasingly jeopardizing the security of cryptosystems currently in widespread use, such as RSA or elliptic-curve signatures. To address this threat, researchers and standardization institutes have accelerated the transition to quantum-resistant cryptosystems, collectively known as Post-Quantum Cryptography (PQC). These PQC schemes present new challenges due to their larger memory and computational footprints and their higher chance of latent vulnerabilities.

In this work, we address these challenges by introducing a scheme to upgrade the digital signatures used by security keys to PQC, focusing on both its theoretical and practical aspects. Specifically, we introduce a hybrid digital signature scheme based on two building blocks: a classically-secure scheme, ECDSA, and a post-quantum secure one, Dilithium. Our hybrid scheme maintains the guarantees of each underlying building block even if the other one is broken, thus being resistant to classical and quantum attacks. Additionally, our hybrid scheme ensures that an adversary cannot derive ECDSA or Dilithium signatures that this authentication protocol considers valid. On the practical aspect, we experimentally show that our hybrid signature scheme can successfully execute on current security keys, even though secure PQC schemes are known to require substantial resources.

We publish an open-source implementation of our scheme at http://anonymous.4open.science/r/OpenSK-D018/[1] so that other researchers can reproduce our results on a nRF52840 development kit.

Keywords: No keywords given.

1 Introduction

Recent advances in quantum computing are increasingly jeopardizing the security of cryptosystems currently in widespread use, such as RSA [42] and DSA [26]. For example, even the comparatively-newer ECDSA, based on elliptic curve cryptography [25], is vulnerable to quantum attacks (i.e., attacks that leverage quantum computers).

This common vulnerability stems from the shared mathematical problems that all these cryptosystems rely upon: factorization[2] and discrete logarithms[3]. These problems are currently considered to be computationally hard for a classical computer but are not quite as impractical for a quantum one: for example, Shor proposed an algorithm to solve factorization and discrete logarithms efficiently in 1994 [43, 39, 44].

[1] The code will be available on Github after acceptance. The code has been anonymized for this review.

[2] Given $N = p \cdot q$ where $p < q, p \approx q$, find p, q.

[3] Given $p, g, g^x \mod p$ find x

Whereas quantum attacks have been purely theoretical, multiple research groups are racing to close the gap between theory and practice. For example, in 2019 Google claimed to have achieved *quantum supremacy* by performing a calculation on a quantum computer that would have taken a classical computer 10,000 years to complete [2], though this is contested [5]. Due to this race, the United States National Institute of Science and Technology (NIST) stated that quantum computing "is a serious long-term threat to the cryptosystems currently standardized by NIST" [13].

given an estimated time frame for the end of this race, stating that it expects RSA-based cryptosystems to be broken in a matter of hours by 2030 [13].

To address this threat, researchers and standardization institutes have accelerated the transition to quantum-attack-resistant cryptosystems, collectively known as Post-Quantum Cryptography (PQC). These PQC schemes rely on a new set of underlying hard problems, e.g. using lattices, that researchers believe to be impervious to quantum attacks. However, these schemes present new challenges due to their substantial memory and computational footprints and their higher chance of latent vulnerabilities due to the schemes' novelty. To mitigate the potential damage, researchers are pursuing hybrid signature schemes [7], which maintain the classical scheme's security against classical attackers, even if researchers find holes in the less-vetted PQC scheme, and vice versa.

Proposing new schemes is a necessary step to disarm the threat that quantum computing poses to cryptosystems in widespread use, but it is not enough. Extensive adoption of these schemes must quickly follow. Among the plethora of protocols that need upgrading to PQC, one class that stands out is security-key-based authentication protocols, such as FIDO's CTAP and WebAuthn [20, 27]. Through these protocols, a user can prove their identity with a hardware token (commonly called a *security key*), either as a second-factor authentication or a first-factor (i.e., passwordless authentication).

In this work, we address these challenges by introducing a PQC digital signature scheme to be used by security keys, focusing on both the theoretical and practical aspects of this scheme. Specifically, we introduce a hybrid digital-signature scheme based on two building blocks: a classically-secure scheme, ECDSA, and a post-quantum secure one, Dilithium. We picked Dilithium [17] as it is one of the schemes recently selected by NIST [34] as the PQC standard for digital signature schemes and because of its fast signing speed. Fast signing is important for our use case since signing is the most frequent operation executed during normal usage of a security key.

Our hybrid scheme maintains the guarantees of each underlying scheme even if the other one is broken, thus being resistant to classical and quantum attacks. In addition, our hybrid scheme offers an additional layer of security by ensuring that an adversary cannot derive stand-alone ECDSA or Dilithium signatures that this authentication protocol considers valid.

On the practical aspect, we show that our hybrid signature scheme can be executed by current security keys, even though secure PQC schemes require substantial resources. In contrast, commercial security keys have little to offer regarding their computational and memory capabilities. Specifically, we implement our hybrid scheme in OpenSK [37], a popular open-source firmware for security keys written in Rust. We provide our implementation as open-source software with an Apache2 license at `http://anonymous.4open.science/r/OpenSK-D018/`.

Our contributions are as follows:

- We improve the security of a previously proposed hybrid signature scheme [7] in the context of security key authentication. On top of maintaining the security guarantees of the underlying classical and post-quantum secure schemes, we provide an additional layer of security tailored for potential downgrade attacks in authentication protocols.
- We release an implementation of our hybrid scheme with ECDSA and Dilithium as underlying components. We have implemented this scheme in Rust on top of the

security-key firmware OpenSK. To allow deployment on diverse hardware, we do not take advantage of any hardware-specific acceleration and we ensure that the memory footprint of our hybrid scheme fits in 64 kB of RAM. This requirement leads us to reduce Dilithium's memory footprint under the same constraint.

1.1 Related Work

Hybrid Cryptosystems. For the transition period to prevalent quantum computers, hybrid cryptosystems provide security against future quantum attackers, while mitigating potential design or implementation bugs in the face of classical attackers. Designing hybrid solutions has been an important line of research for multiple cryptosystems in use today, such as authenticated key exchange [16, 3], public-key encryption [31], and digital signatures [7].

In terms of hybrid signature scheme design, the work of Bindel et al. [7] is the most similar to ours. The main difference between our work and [7] is that we also explore the practical considerations of implementing hybrid signature schemes under the constraints of embedded hardware. In addition, we refine the non-separability property they introduce, and we extend it to a more general class of hybrid signature schemes.

Dilithium vs other PQC. Dilithium [17] and Falcon [30] both won NIST's post-quantum signature algorithm standardization. We compare the two schemes in Figure 1. As will be discussed in Section 5, for security keys, we are mainly interested in the signing speed and the private key size. Signing is the most common operation for security keys, as verification of its produced signatures is not performed on embedded hardware. The private key size affects how many credentials can be stored on the security key.

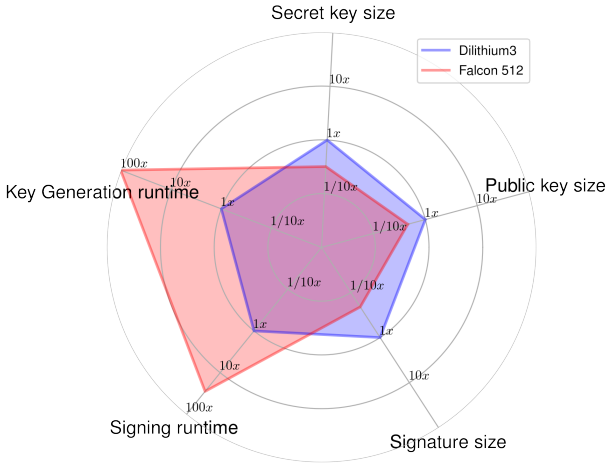


Figure 1: Relative performance of Falcon 512 compared to the reference implementation of Dilithium3 as the baseline for the security key use case, as reported by Raavi et al. [40] in Figure 6.

We optimize Dilithium in Section 5 to get closer to Falcon in key sizes. Our implementation maintains competitive signing speeds, while having favorable properties overall:

- *Private key size:* We store the 256-bit secret that we use to regenerate the private key on the fly. Given the amount of entropy we need, this is near-optimal in terms of required storage.

- *Public key and signature size:* Public keys and signatures have to be small enough for transmission over USB and NFC. Falcon has 2x smaller public keys and 3x smaller signatures than Dilithium for comparable security levels. Dilithium is still compatible with constraints from the CTAP protocol.
- *Key generation speed:* Dilithium is 100x faster for key generation [23] than Falcon. That is why we can regenerate private keys on the fly from the stored random seed.
- *Memory:* Falcon has smaller requirements, but Dilithium can be optimized to fit embedded devices [8].

Hash-based signatures are another interesting candidate that provide different benefits over lattice-based signatures, in particular a very small key size. XMSS [12] and LMS [28] are two stateful hash-based signatures, and while the use case of authentication against a server generally allows stateful interactions, it still increases protocol and implementation complexity. In particular, this use case would require state management on the security key itself. There are stateless hash-based signature schemes like SPHINCS [4], and SPHINCS+ [5] which has been selected as a standard by NIST. However, these have a much larger signature size that is infeasible for our use case, and the performance cost of signing compared to lattice schemes is significantly worse.

Improvements to Dilithium's performance. Multiple works have focused on improving Dilithium's reference implementation [38] in terms of speed, memory footprint, or security against side-channel attacks. Future Dilithium optimizations will also compliment our own implementation and improve the user experience on security keys with faster runtimes.

As Dilithium is based on the Fiat Shamir with Aborts framework, the work of Ravi et al. [41] focuses on early detection of rejected samples, improving the speed of Dilithium's signing procedure. Their proposed optimizations are also evaluated on the ARM Cortex-M4F MCU. Abdulrahman et al. focused on further improvements of Dilithium's speed on the ARM Cortex-M4 [1].

Greconici et al. [22] obtain a constant-time Dilithium implementation on the Cortex-M3, which is necessary for limiting potential side-channel attacks. In addition, they present different strategies for the signing procedure that allow trading between the stack and flash memory usage and speed, which can be applied for Cortex-M3 and Cortex-M4. The recent work of Bos et al. [8] focuses on reducing Dilithium's memory footprint to less than 9kB.

Migliore et al. [33] analyze Dilithium's vulnerabilities against side-channels attacks when implemented on an ARM Cortex-M3 micro-controller, and remove unexpected leakages through masking.

2 Background

In this section, we introduce the relevant cryptography, describe our use case of security keys and explain our hardware and firmware stack for embedded development.

2.1 Digital Signatures

We first recall the definition of a digital signature scheme. For our scope, we only consider signature schemes deployed on classical computers.

Definition 1. (Digital signature scheme) A digital signature scheme Σ is a triple of polynomial time algorithms $(\Sigma.\texttt{KeyGen}, \Sigma.\texttt{Sign}, \Sigma.\texttt{Verify})$ such that:

- $\Sigma.\texttt{KeyGen}(1^\kappa)$ is a probabilistic algorithm that takes the security parameter 1^κ as input and outputs a public verification key $\texttt{pk}$ and a secret signing key $\texttt{sk}$.

- $\Sigma.\mathsf{Sign}(m, \mathsf{sk})$ is a probabilistic algorithm that takes a message m and a secret key sk as inputs and outputs a signature σ.
- $\Sigma.\mathsf{Verify}(m, \sigma, \mathsf{pk})$ is a deterministic algorithm that takes a message m, a signature σ and a public key pk as inputs. It outputs true or false, where true means that σ is accepted as a signature for the message m and public key pk, and false means that the signature is not accepted.

Digital signature schemes must achieve two properties: *correctness* and *security*. Correctness requires that for every key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow_\$ \Sigma.\mathsf{KeyGen}(1^\kappa)$, every possible message m, and any possible $\sigma \leftarrow_\$ \Sigma.\mathsf{Sign}(m, \mathsf{sk})$, it holds that $\Sigma.\mathsf{Verify}(m, \sigma, \mathsf{pk}) = \mathsf{true}$. In terms of security, there are multiple definitions, and we present the ones relevant for our context in the subsection below.

The security of digital signatures is often defined through a security game where an adversary tries to forge a valid signature while interacting with a challenger who holds the secret key. For the scope of our paper, we are only interested in the transition to post-quantum digital signatures. That is, we assume that the signature schemes are only executed on classical computers, hence the adversary (quantum or classical) only interacts with a classical challenger or signing oracle. We will use the notation C-adversary to refer to a classical adversary, and Q-adversary to refer to a quantum adversary with classical access to the signing oracle.

Security guarantees for digital signatures are often defined through the capabilities of the adversary (i.e. whether it has access to a quantum computer for its local computations, or whether it can adaptively choose the messages it queries signatures for), and the constraints regarding the signature the adversary has to forge to win the security game (i.e. the signed message may be given in advance by the challenger, or may be chosen by the adversary as long as the message was not already signed by the challenger). We only work with two security definitions, defined below: EUF-CMA (Existential Unforgeability under Chosen Message Attacks) and SUF-CMA (Strong Unforgeability under Chosen Message Attacks).

Definition 2. (EUF-CMA security) We consider the EUF-CMA security game for a signature scheme Σ, where the adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$ as follows:

1. The challenger obtains a pair of keys $(\mathsf{sk}, \mathsf{pk}) \leftarrow_\$ \Sigma.\mathsf{KeyGen}(1^\kappa)$ and sends pk to $\mathcal{A}$.
2. $\mathcal{A}$ may adaptively send a polynomial number ($\mathsf{poly}(\kappa)$, where κ denotes the security parameter) of queries m_i to the challenger $\mathcal{C}$. For each such query, $\mathcal{C}$ obtains $\sigma_i = \Sigma.\mathsf{Sign}(m, \mathsf{sk})$ and sends σ_i to the adversary. Note that $\mathcal{A}$ may send query m_{i+1} after receiving σ_i.
3. $\mathcal{A}$ may send a message-signature pair $(m\ , \sigma\ )$. $\mathcal{A}$ wins the EUF-CMA security game if $m\ \ /\ \{$query $m_i\}$ and $\Sigma.\mathsf{Verify}(m\ , \sigma\ , \mathsf{pk})$ holds.

We say that Σ is C-EUF-CMA secure if any classical $\mathcal{A}$ wins the EUF-CMA security game with negligible probability ($\mathsf{negl}(\kappa)$). Similarly, Σ is Q-EUF-CMA secure if any (possibly quantum) $\mathcal{A}$ that interacts with the signing oracle classically wins the EUF-CMA security game with negligible probability ($\mathsf{negl}(\kappa)$).

Definition 3. (SUF-CMA security) We consider the SUF-CMA security game for a signature scheme Σ, where the adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$ as follows:

1. The challenger obtains a pair of keys $(\mathsf{sk}, \mathsf{pk}) \leftarrow_\$ \Sigma.\mathsf{KeyGen}(1^\kappa)$ and sends pk to $\mathcal{A}$.
2. $\mathcal{A}$ may adaptively send a polynomial number ($\mathsf{poly}(\kappa)$) of queries m_i to the challenger $\mathcal{C}$. For each such query, $\mathcal{C}$ obtains $\sigma_i = \Sigma.\mathsf{Sign}(m, \mathsf{sk})$ and sends σ_i to the adversary. Note that $\mathcal{A}$ may send query m_{i+1} after receiving σ_i.

3. $\mathcal{A}$ may send a message-signature pair (m, σ). $\mathcal{A}$ wins the SUF-CMA security game if $(m, \sigma) \notin \{(m_i, \sigma_i) \mid m_i \text{ query}\}$ and $\Sigma.\mathsf{Verify}(m, \sigma, \mathsf{pk})$ holds.

We say that Σ is C-SUF-CMA secure if any classical $\mathcal{A}$ wins the SUF-CMA security game with negligible probability ($\mathsf{negl}(\kappa)$). Similarly, Σ is Q-SUF-CMA secure if any (possibly quantum) $\mathcal{A}$ that interacts with the signing oracle classically wins the SUF-CMA security game with negligible probability ($\mathsf{negl}(\kappa)$).

We recall the security guarantees of the concrete digital signature schemes that we use:

- ECDSA achieves C-EUF-CMA security in the random bijection model [19]. ECDSA's security has also been studied in the generic group model [11, 10]. However, the analysis in the generic group model establishes that ECDSA achieves even C-SUF-CMA security, while in fact it is trivial to find a new signature for an already signed message [19].

- Dilithium achieves Q-SUF-CMA security in the quantum random oracle model [24] (which implies Q-EUF-CMA).

2.2 Post-quantum cryptography

Dilithium is a signature scheme without known weaknesses to quantum computers. Different parameters sets of Dilithium are called modes and correspond to estimated security levels. The cryptographic strengths of Dilithium are shown in Table 1. NIST made an attempt to translate these hardness levels to classical cryptography [35]. While classical and quantum security levels are hard to directly compare, we add these estimates to the table as an approximation.

Table 1: Cryptographic strength of Dilithium modes, as of NIST standardization round 3 (see [14], Table 1). Numbers in parenthesis refer to SUF-CMA. The classical equivalent refers to NIST's estimation.

Mode	LWE Hardness	SIS Hardness (for SUF-CMA)	Estimated classical equivalent
Dilithium2	112	112 (110)	collision in SHA256
Dilithium3	165	169 (159)	key search in AES-192
Dilithium5	229	241 (230)	key search in AES-256

2.3 Security Keys

Security keys allow user authentication with digital signatures. They are often implemented on embedded hardware to protect secret key material from extraction.

FIDO. Fast IDentity Online (*FIDO*, see [20]) is set of standards to allow online authentication through asymmetric cryptography. This exchange of messages involves two protocols: the Client to Authenticator Protocol (*CTAP*, see [15]), which enables the communication between the user's *Authenticator* and their *Client* (such as their browser, or their computer), and WebAuthn, which ensures the communication between the client and the server (*Relying Party*). Security keys act as authenticators and therefore implement CTAP.

CTAP. As part of a CTAP registration, the user generates a key pair, and sends the public key to the server. For a CTAP authentication, the user then proves possession of the private key (*Credential*) being stored on an authenticator. A credential can be stored in one of two ways: Either it is encrypted and sent to the relying party for storage, or it is stored locally in flash. We call these cases server-side key and resident key, respectively.

We describe the cryptographic commands in CTAP below (see Figure 2). The CTAP protocol started with U2F [27], and since then evolved to its current version 2.1. The most important commands are Make Credential for registration and Get Assertion for authentication. Depending on usage of server-side or resident credentials, these commands use the following cryptographic operations:

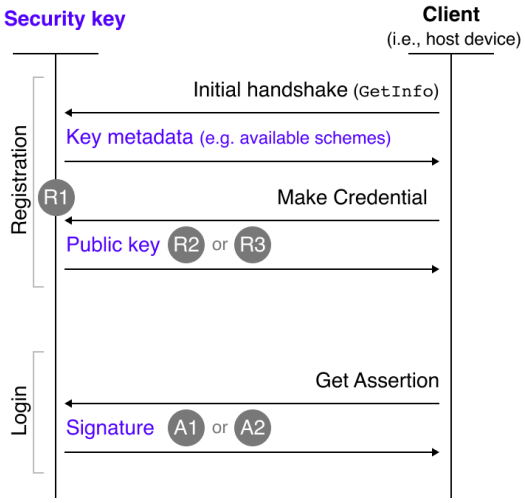
R1) During registration, the security key generates a key pair.

R2) Registration returns the public key of the credential, and may return the encrypted private key (server-side key).

R3) Registration returns the public key of the credential, and may store the private key on flash (resident key).

A1) Authentication returns a signature over a response derived from the Relying Party's message.

A2) Authentication returns a signature, and may return an encrypted private key (server-side vs resident key).



(a)

Figure 2: Cryptographic operations in the CTAP protocol.

3 Attacker model

Security keys' main goal is to defend against remote attackers and phishing. Defense against local attackers with physical possession of the device are an explicit non-goal. Attackers can attack the protocol on different levels: cryptographically, on CTAP level, or against the hardware device. In our cryptography analysis in Section 4, we consider different extended capabilities for attackers:

- Possession of a Cryptographically-Relevant Quantum Computer;
- Knowledge of a Dilithium weakness;
- Downgrade of the cryptographic protocol.

We acknowledge that FIDO's protocols mitigate downgrading the protocol already. They transmit the used algorithm over a channel that is considered secure in their attacker model. On the other hand, real-life adversaries are often able to find and exploit vulnerabilities in implementations. Therefore, we consider it good cryptographic design to consider this attacker capability regardless and to enhance the guarantees of the hardware security key accordingly.

Cryptographic strength. We want our implementation to support all modes of Dilithium, to allow applications with strong security requirements. In particular, security keys are an important line of defense against account hijacking.

Non-goals. Local attacks against the hardware itself or faulty implementations are out of scope for this work. That includes local side-channel attacks. Indeed, Dilithium has been successfully attacked locally on e.g. the power side-channel [32]. We follow FIDO's security assumptions, listed in their Security Reference [21]. The two most important for our threat model are the following:

SA-3 Applications on the user device are able to establish secure channels that provide trustworthy server authentication, and confidentiality and integrity for messages (e.g., through TLS).

SA-4 The computing environment on the FIDO user device and the applications involved in a FIDO operation act as trustworthy agents of the user.

As mentioned above, this generally makes downgrade attacks impossible. However, the Security Reference mentions hardening against "Protocol level real-time MITM attack", motivating our mitigations against hybrid signature downgradability as described in Section 4.2.

4 Hybrid Signatures

A hybrid signature scheme combines a classical signature algorithm with a post-quantum secure signature algorithm. Before discussing the design of our hybrid scheme, we explain why such an approach is relevant instead of simply replacing classically secure schemes with post-quantum secure schemes. We present the assumptions below:

1. Cryptographically-Relevant Quantum Computers (i.e. with enough qubits to break ECDSA) are not available yet.
2. Classical signature algorithms withstands attacks from classical computers.
3. The post-quantum secure signature algorithm might be breakable by classical computers due to design or implementation bugs.

If any of these assumptions fails, using a hybrid approach instead of replacing classical schemes with post-quantum schemes indeed does not add any security. We believe that all of these assumptions are currently correct. The third assumption is motivated by a newly discovered attack against Rainbow [6], one of the NIST standardization finalists.

We can now discuss the informal requirements a hybrid scheme H should satisfy:

1. If a quantum computer becomes available, and hence H's underlying classical scheme is broken, H should maintain the security of its underlying post-quantum scheme.

2. If a classical attack for H's underlying post-quantum secure scheme is discovered, H should maintain the security of its underlying classical scheme.

There are multiple natural options for designing a hybrid scheme that satisfies such guarantees. An example is obtaining a hybrid signature by concatenating a classical signature with a post-quantum secure signature. Although simple, this approach indeed maintains existential unforgeability of the underlying schemes [7].

On the other hand, for our concrete instantiation, Dilithium is strongly unforgeable, while ECDSA is existentially but not strongly unforgeable [19]. Concatenation unfortunately would not maintain Dilithium's strong unforgeability: one could simply replace the ECDSA part of the hybrid signature with another valid ECDSA signature.

Intuitively, this issue can be solved by first signing a given message m with the X-EUF-CMA secure scheme, obtaining σ_1, and afterwards obtaining σ_2 by signing (m, σ_1) with the X-SUF-CMA secure scheme. The hybrid signature for the message m is $\sigma = (\sigma_1, \sigma_2)$. This approach is called *Strong Nesting* [7] and is the basis of our hybrid scheme.

Replacing ECDSA with Ed25519 is another possible fix, as the latter is strongly unforgeable [9]. However, as all security keys already implement ECDSA, a hybrid protocol that uses ECDSA benefits from code reuse.

Strong Nesting still has a potential caveat. As mentioned in [7], it leaks valid signatures for one of its underlying schemes from the hybrid counterpart. This could pose an issue in our concrete use case of the hybrid scheme: in theory, an adversary performing a man-in-the-middle attack between the client and the relying party could intercept the hybrid public keys and signatures sent by the client to the relying party, and only forward the ECDSA components instead. This way, the user would authenticate through a classically secure scheme, while believing they would be using a post-quantum secure scheme. Although CTAP offers protection against such downgrade attacks, we would like our hybrid scheme to offer an additional layer of security. Therefore, we add the following informal requirements:

3. It should be difficult for a classical adversary to derive valid classical components of the hybrid signature that are relevant for CTAP.

4. It should be difficult for a quantum adversary to derive valid post-quantum components of the hybrid signature that are relevant for CTAP.

To obtain this additional layer of security, we build upon the *non-separability* property introduced in [7], which roughly prevents an adversary from deriving components of the hybrid signature (*partial* signatures) that could pass as valid signature coming from one of the underlying schemes. We formally define this property in Section 4.2. We mention that preventing an adversary from deriving such components is impossible as long as the hybrid scheme's verification algorithm explicitly uses the underlying scheme's verification algorithm. Fortunately, we can prevent the adversary from deriving partial signatures for messages that are useful in CTAP.

We make use of the suggestion of Bindel et al. [7] of prepending a label to the message to be signed. This additions restricts an adversary from trivially deriving partial signatures for messages that have the chosen label as a prefix, if the underlying schemes are secure.

In order to choose the right label, we note that messages in CTAP follow a strict format: each relying party RP with identifier $\texttt{id}_{RP}$ requests signatures for messages with prefix $\texttt{SHA256}(\texttt{id}_{RP})$. Then, for our scope, it is enough to prevent the adversary from deriving messages whose first 32 bytes represent the hashed identifier of some relying party. We choose a 32 B label `label` uniformly at random from SHA256's output space $\{0,1\}^{256}$, meaning that an adversary would only be able to use a derived signature in CTAP if the relying party it interacts with has $\texttt{SHA256}(\texttt{id}_{RP}) = \texttt{label}$. To find a relying party

identifier that hashes to our label, an attacker would need to generate a SHA256 preimage, while SHA256 is believed to be hard to invert.

We only generate the label once and then we treat it as a constant; it is hardcoded in every implementation of our scheme. That is, the label is the same for every party using the scheme. As an alternative construction with the same properties, one could chose a random label during key generation and store it in both the secret and public key. We decided to use the constant label approach because it leads to smaller key size.

We can now formally present our hybrid signature scheme. Given two signature schemes Σ_1 and Σ_2, we define the secret and public keys as pairs of their counterparts in the given underlying schemes. Below we present the pseudocode of the KeyGen() function.

$\underline{\boldsymbol{H}(\Sigma_1, \Sigma_2).\mathtt{KeyGen}(1^\kappa)}$

$\mathsf{sk}_1, \mathsf{pk}_1 \leftarrow\!\!\$\ \Sigma_1.\mathtt{KeyGen}(1^\kappa)$

$\mathsf{sk}_2, \mathsf{pk}_2 \leftarrow\!\!\$\ \Sigma_2.\mathtt{KeyGen}(1^\kappa)$

return $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2), \mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2)$

When signing a message m, the signer first obtains m' by prepending the hardcoded label to the message m. Then, it obtains a Σ_1-signature for m', followed by a Σ_2-signature for (m', σ_1). The hybrid signature is then the pair (σ_1, σ_2). In practice, these pairs can be implemented as a simple concatenation if σ_1 has predictable length, or a concatenation with a separator. The pseudocode of the signing and verifying functions is presented below.

$\underline{\boldsymbol{H}(\Sigma_1, \Sigma_2).\mathtt{Sign}(m, \mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2))}$

$m' = \mathtt{label} \| m$

$\sigma_1 \leftarrow \Sigma_1.\mathtt{Sign}(m', \mathsf{sk}_1)$

$\sigma_2 \leftarrow \Sigma_2.\mathtt{Sign}((m', \sigma_1), \mathsf{sk}_2)$

return $\sigma = (\sigma_1, \sigma_2)$

$\underline{\boldsymbol{H}(\Sigma_1, \Sigma_2).\mathtt{Verify}(m, \sigma, \mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2))}$

$m' = \mathtt{label} \| m$

obtain σ_1 and σ_2 from σ

return $(\Sigma_1.\mathtt{Verify}(m', \sigma_1, \mathsf{pk}_1) \wedge$
$\Sigma_2.\mathtt{Verify}((m', \sigma_1), \sigma_2, \mathsf{pk}_2))$

We first show that our proposed hybrid scheme $\boldsymbol{H}(\Sigma_1, \Sigma_2)$ achieves correctness.

Lemma 1. *If Σ_1 and Σ_2 are correct, then $\boldsymbol{H}(\Sigma_1, \Sigma_2)$ is also correct.*

Proof. Let $\big(\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2), \mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2)\big)$ denote an arbitrary key pair that can be obtained from $\boldsymbol{H}(\Sigma_1, \Sigma_2).\mathtt{Sign}(m, \mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2))$ with probability greater than zero.

Let m denote an arbitrary message, and let $\sigma = (\sigma_1, \sigma_2)$ denote an arbitrary signature that $\boldsymbol{H}(\Sigma_1, \Sigma_2).\mathtt{Sign}(m, \mathsf{sk})$ can output with non-zero probability. Since σ_1 is a Σ_1-signature obtained by signing $\mathtt{label}\ m$ using sk_1 and Σ_1 achieves correctness, $\Sigma_1.\mathtt{Verify}(\mathtt{label}\ m, \sigma_1, \mathsf{pk}_1)$ holds. Similarly, $\Sigma_2.\mathtt{Verify}((\mathtt{label}\ m, \sigma_1), \sigma_2, \mathsf{pk}_2))$ holds. Hence, the $\boldsymbol{H}(\Sigma_1, \Sigma_2).\mathtt{Verify}(m, \sigma, \mathsf{pk}) = \mathsf{true}$, which means that our hybrid scheme is correct. □

4.1 Security Analysis

In this subsection, we show that our hybrid scheme $\boldsymbol{H}(\Sigma_1, \Sigma_2)$ maintains the security guarantees of its underlying components. In the statements, we use $X \quad \{C, Q\}$ to specify whether we consider classical or post-quantum security. As our formal proofs only employ standard techniques, we include them in the appendix for completeness.

Lemma 2. *If Σ_1 is X-EUF-CMA secure, then $\boldsymbol{H}(\Sigma_1, \Sigma_2)$ is X-EUF-CMA secure as well.*

Proof Sketch. If an X-adversary $\boldsymbol{A}$ wins the X-EUF-CMA security game for $\boldsymbol{H}(\Sigma_1, \Sigma_2)$, then it outputs a pair $(m', (\sigma_1, \sigma_2))$ such that: it has never queried a signature for the message m', σ_1 is a valid Σ_1-signature for $\mathtt{label}\ m'$, and σ_2 is a valid Σ_2-signature for

$(\texttt{label}\ m\ , \sigma_1)$. Therefore, A has to produce a Σ_1-signature, and hence one can use A to create an adversary that breaks Σ_1's X-EUF-CMA security.

□

Lemma 3. *If Σ_2 is X-EUF-CMA secure (resp. X-SUF-CMA) secure, $H(\Sigma_1, \Sigma_2)$ is X-EUF-CMA secure (resp. X-SUF-CMA) as well.*

Proof Sketch. Similarly to the previous proof sketch, if an X-adversary A wins the X-EUF-CMA for $H(\Sigma_1, \Sigma_2)$, then it outputs a pair $(m\ , (\sigma_1, \sigma_2))$ such that: it has never queried a signature for the message m , σ_1 is a valid Σ_1-signature for $\texttt{label}\ m$, and σ_2 is a valid Σ_2-signature for $(\texttt{label}\ m\ , \sigma_1)$.

Using A, one can construct an adversary B that wins the X-EUF-CMA security game of Σ_2 with at least the same success probability as A. Then, as Σ_2 is existentially unforgeable, this implies that the success probability of any A for $H(\Sigma_1, \Sigma_2)$ is negligible, and hence $H(\Sigma_1, \Sigma_2)$ is also existentially unforgeable.

For X-SUF-CMA security, we need to take into account the fact that A might have obtained a new signature (σ_1, σ_2) for a previously queried m . We need to show how B can derive its own forgery for Σ_2's X-SUF-CMA security game in this case, since the argument is otherwise identical. For each query m_i sent by A, B obtains a Σ_1-signature for $\texttt{label}\ m_i$ by itself, denoted by $\sigma_{1,i}$, and sends $(\texttt{label}\ m_i, \sigma_{1,i})$ to its challenger to obtain the corresponding Σ_2-signature $\sigma_{2,i}$. If there is some query such that $\big((\texttt{label}\ m_i, \sigma_{1,i}), \sigma_{2,i}\big) = \big((\texttt{label}\ m\ , \sigma_1), \sigma_2\big)$, then $m_i = m$, $\sigma_{1,i} = \sigma_1$, and $\sigma_{2,i} = \sigma_2$, which means that A's forgery was not successful. Hence, $\big((\texttt{label}\ m\ , \sigma_1), \sigma_2\big)$ is a successful forgery for B. □

4.2 Non-Separability

Our hybrid scheme $H(\Sigma_1, \Sigma_2)$ offers an additional layer of security for downgrade attacks. To achieve this, we build upon the *non-separability* property introduced in [7], which intuitively prevents an adversary from obtaining *valid* and *useful* Σ_t-signatures (for a chosen t $\{1, 2\}$) from a hybrid scheme $H(\Sigma_1, \Sigma_2)$.

Firstly, we explain the term *valid*. The work of Bindel et al. considers hybrid schemes in which a single pair of Σ_t-keys is obtained, namely in the key generation algorithm. Naturally, the adversary is expected not to be able to derive useful message-signature pairs valid with respect to these Σ_t-keys. We extend the definition for a more general class of hybrid schemes, in which Σ_t-keys may be generated in the hybrid key generation and/or when signing (similarly to constructions based on one-time signatures [18]). The adversary is expected not to derive meaningful message-signature pairs that can be verified using any Σ_t-public key generated by the signer at any time.

Secondly, we explain the term *useful*. As mentioned previously, it is impossible to prevent the adversary from obtaining valid Σ_t components from the hybrid scheme if the verification algorithm of the hybrid scheme uses Σ_t's verification algorithm explicitly. In this case, non-separability can only prevent the adversary from obtaining *useful* Σ_t components. The term useful depends strongly on the context and is defined with respect to a set of messages U M_{Σ_t}, where M_{Σ_t} denotes the message space of the signature scheme Σ_t. The set U becomes a parameter of the non-separability property: informally, non-separability with respect to U prevents an adversary from deriving Σ_t-signatures for messages m U from the hybrid scheme, and this property is useful if the messages in $M_{\Sigma_t} \setminus U$ are meaningless for the practical context. For our scope, we will define $U_{\texttt{label}} := \{m \mid m \text{ does not have prefix } \texttt{label}\}$, as the probability that a message with prefix `label` needs to be signed in the context of CTAP is small.

We stress that non-separability only refers to one of the hybrid scheme's underlying schemes: a hybrid scheme $H(\Sigma_1, \Sigma_2)$ can guarantee that it is difficult for an adversary to derive useful signatures for Σ_1, while it might be easy to derive useful Σ_2 components of

the signature. Hence, in addition to the useful set of messages U, the definition is also parametrized by $t \in \{1, 2\}$, denoting the index of the underlying signature scheme that should be hard to separate, and $X \in \{C, Q\}$, denoting the type of adversary for whom it should be difficult to obtain partial Σ_t-signatures.

Then, given $t \in \{1, 2\}$, $X \in \{C, Q\}$ and U, we define (t, X, U)-non-separability property for a hybrid scheme $H(\Sigma_1, \Sigma_2)$ as follows:

Definition 4. ((t, X, U)-Non-separability) We consider the (t, X, U)-non-separability security game for $H(\Sigma_1, \Sigma_2)$, where an X-adversary A interacts with a challenger C_t as follows:

1. C_t generates a pair of keys $(\mathsf{sk}, \mathsf{pk}) \leftarrow_\$ H(\Sigma_1, \Sigma_2).\mathsf{KeyGen}(1^\kappa)$. C_Σ saves any Σ_t-public keys obtained in the key generation algorithm in a set P_t. Then, C_t sends pk to A.
2. A may send a polynomial number ($\mathsf{poly}(\kappa)$) of message queries m_i to C_t adaptively. For each query m_i, C_t computes the signature $\sigma_i \leftarrow_\$ H(\Sigma_1, \Sigma_2).\mathsf{Sign}(m_i, \mathsf{sk})$ and saves the Σ_t-public keys generated by the signature algorithm in a set P_{m_i}. Then, C_t sets $\mathsf{P}_t := \mathsf{P}_t \cup \mathsf{P}_{m_i}$ and sends $(\sigma_i, \mathsf{P}_{m_i})$ to A.
3. At some point, A sends a tuple (m, σ, pk) to C_t.

A wins the game if each of the following conditions holds:

- the public key was used in the hybrid signature at some point in the game: $\mathsf{pk} \in \mathsf{P}_t$;
- the signature is valid: $\Sigma_t.\mathsf{Verify}(m, \sigma, \mathsf{pk}) = \mathsf{true}$;
- the message is useful: $m \in \mathsf{U}$.

Then, $H(\Sigma_1, \Sigma_2)$ is (t, X, U)-non-separable if any X-adversary wins the (t, X, U)-non-separability security game with probability negligible in the security parameter κ.

The results below show that our hybrid scheme $\mathsf{H}(\Sigma_1, \Sigma_2)$ guarantees non-separability for both Σ_1 and Σ_2 as long as the two underlying schemes provide security. Similarly to the previous results, our statements use $X \in \{C, Q\}$ to specify whether we consider classical or post-quantum security. The formal proofs are included in the appendix.

Lemma 4. *Let $\mathsf{U}_{\mathtt{label}}$ denote the set $\{m \mid m$ does not have prefix $\mathtt{label}\}$. If Σ_1 is X-EUF-CMA secure, $\mathsf{H}(\Sigma_1, \Sigma_2)$ is $(1, X, \mathsf{U}_{\mathtt{label}})$-non-separable.*

Proof Sketch. Since $\mathsf{H}(\Sigma_1, \Sigma_2)$ prepends $\mathtt{label}$ to any given message m before signing it using Σ_1, an X-adversary A only observes Σ_1-signatures for messages that are not in $\mathsf{U}_{\mathtt{label}}$, hence have prefix $\mathtt{label}$, while it has to output a valid Σ_1 signature for a message m that does not have the prefix $\mathtt{label}$.

One can then use A to construct an adversary B for Σ_1's X-EUF-CMA security game that simulates the non-separability game perfectly towards A, and wins at least with the same success probability as A. The winning probability is maintained as whenever A outputs a successful tuple $(m, \sigma_1, \mathsf{pk})$, B can simply forward (m, σ_1) to its challenger and win as well since B only needs to query signatures for messages with the prefix label.

Then, as Σ_1 is X-EUF-CMA secure, the success probability of every X-adversary in Σ_1's X-EUF-CMA security game is negligible, hence every X-adversary has negligible success probability in winning the $(1, X, \mathsf{U}_{\mathtt{label}})$-non-separability game for $\mathsf{H}(\Sigma_1, \Sigma_2)$. □

As the lemma below can be proven analogously to Lemma 4, we omit the proof sketch.

Lemma 5. *Let $\mathsf{U}_{\mathtt{label}}$ denote the set $\{m \mid m$ does not have prefix $\mathtt{label}\}$. If Σ_2 is X-EUF-CMA secure, $\mathsf{H}(\Sigma_1, \Sigma_2)$ is $(2, X, \mathsf{U}_{\mathtt{label}})$-non-separable.*

4.3 Final guarantees

The next theorem follows immediately from the lemmas proven in the previous subsections.

Theorem 1. *$H(\Sigma_1, \Sigma_2)$ offers the following guarantees:*

1. *If Σ_1 is X-EUF-CMA secure, then $H(\Sigma_1, \Sigma_2)$ is also X-EUF-CMA secure and $(1, X, U_{\texttt{label}})$-non-separable for $U_{\texttt{label}} = \{m \mid m$ does not have prefix $\texttt{label}\}$.*
2. *If Σ_2 is X-EUF-CMA (resp. X-SUF-CMA) secure, then $H(\Sigma_1, \Sigma_2)$ is also X-EUF-CMA (resp. X-SUF-CMA) secure and $(2, X, U_{\texttt{label}})$-non-separable for $U_{\texttt{label}} = \{m \mid m$ does not have prefix $\texttt{label}\}$.*

We recall that ECDSA achieves C-EUF-CMA security in the random bijection model, while Dilithium achieves Q-SUF-CMA security in the quantum random oracle model, and we note that our hybrid scheme and proofs use the underlying components as black-boxes. Then, Theorem 1 shows that our scheme $H(\text{ECDSA}, \text{Dilithium})$ at least maintains the security guarantees of ECDSA and Dilithium (in their corresponding models). In addition, $H(\text{ECDSA}, \text{Dilithium})$ provides an additional layer of security for downgrade attacks, as it prevents the adversary for deriving ECDSA or Dilithium signatures for messages that relevant in CTAP.

5 A SK-friendly Implementation

Security keys often run on embedded hardware devices with tight performance constraints. Our work is based on the open source security key *OpenSK* [37]. OpenSK is a firmware that implements CTAP 2.1. It works as an application on top of the embedded operating system TockOS [29]. For this work, we run OpenSK on a Nordic nRF52840 development kit [36] with a 64 MHz ARM Cortex-M4F MCU. The nRF52840 comes with a TRNG for randomness, and we run all CTAP communication over USB.

To support different hardware targets, we want our firmware including Dilithium, namely the key generation and signing algorithm, to fit 64 kB of RAM. For embedded hardware, we discuss various trade-offs between speed, memory usage and key sizes. We describe our changes to Dilithium compared to the reference implementation. We focus on obtaining a hardware security key-friendly Dilithium implementation for all Dilithium modes.

5.1 CTAP requirements

Time to login affects usability. In addition, there are some hard limits for FIDO operations in the specification:

- User presence may timeout after 10 seconds (see 5. Terminology in [15]).
- The size of a CTAP message over USB cannot exceed 7609 B (see 11.2.4. Message and packet structure in [15]).

Following the command naming from Section 2.3, this yields the following priorities:

R1 Key generation must finish in less than 10 s.

R2 Key pairs must be smaller than 7 kB.

R3 The private key should be small to allow storing additional credentials.

A1 The login operation is more frequent than registration. Signing should be as fast as possible.

A2 A private key and signature together must be smaller than 7 kB.

For the reference implementation, the Dilithium modes achieving the desired security requirements (Dilithium3 and Dilithium5) fail some requirements due to the large sizes of the key pair and signatures. Namely, requirement R3 is broken by Dilithium5, while requirements R3 and A2 are broken by both Dilithium3 and Dilithium5.

In the following sections, we will achieve these requirements within the memory limits of embedded hardware. We focus on reducing the private key size significantly. See the experiments section for speed benchmarks.

5.2 Dilithium Optimizations

Our implementation offers two modes: First, a high speed mode, which follows the original implementation with the exception that we reduce the key size. Second, a low memory footprint mode. To reduce Dilithium's memory footprint, we used some of the tricks from [8]. We recompute some intermediate values and effectively trade additional computations (performance) to reduce memory usage.

Both the key generation and the signing algorithm of Dilithium require computations on vectors and matrices of polynomials stored on the stack memory. Intermediate results are also stored on the stack memory. The signing algorithm of reference implementation of Dilithium [38] keeps on stack 49 such polynomials for Dilithium2, 76 for Dilithium3, and 118 for Dilithium5. Each such polynomial requires 1 kB. The secret key and the array of bytes used to compute the signature are stored on the stack at the same time, which leads to a stack usage of at least 53 kB for Dilithium2, 83 kB for Dilithium3, and 127 kB for Dilithium5. In addition, moving the declarations of data structures into scopes, so they are only stored when necessary, would not make a significant difference: the minimum stack usage becomes 40 kB, 66 kB, and 133 kB respectively. The reference implementation of Dilithium is therefore infeasible for our RAM target, since we aim for the security levels of Dilithium3 and Dilithium5.

Fortunately, the computations on polynomials are done sequentially (polynomial by polynomial), and not in parallel, which enables us to only store a few polynomials at a time in the stack memory instead of a significant number of large structures. Sequential execution is appropriate here since OpenSK does not have parallel execution. In addition, we take into account the life cycle of variables and we arrange the code into multiple blocks, such that the polynomials are only stored on the stack when needed, and afterwards the memory can be recycled.

While using this approach reduces the stack usage, it requires some of the intermediate results to be recomputed, and hence it increases the runtime significantly.

Discarding information from the secret key. Dilithium's secret key is an array of bytes comprising the encoding of an array of polynomials, t_0, and the information necessary for computing the array t_0. At a high level, from the secret key, one can derive a matrix of polynomials, A, and two vectors of polynomials, s_1 and s_2. The array t_0 is obtained by reducing each coefficient of $t = A \cdot s_1 + s_2$ modulo 2^d, where d is a parameter. Then, storing the encodings of t_0 is not necessary. To further decrease Dilithium's memory footprint, we can simply recompute these polynomials when signing instead.

Encoding a single polynomial of t_0 into the secret key takes 416 B. In the case of Dilithium2, the encoding of t_0 requires 1664 B. For Dilithium3 and Dilithium5, the encoding requires 2496 B and 3328 B respectively. Then, the size of the secret key gets reduced significantly: from 2528 B to 864 B in the case of Dilithium2, from 4000 B to 1504 B in Dilithium3, and from 4864 B to 1536 B in the case of Dilithium5.

Recomputing t_0 every time we sign helps us decrease Dilithium's memory footprint even more, with the caveat that we need to recompute t_0 every time a message is signed.

Indeed, this change negatively affects the performance of Dilithium, but it remains reasonable (see experiments in Section 6).

Only storing a 32 B seed. Dilithium's key generation uses a 32 B seed as source of randomness, which is then expanded to compute the components of the secret and public keys. We can store only this seed and recompute the secret key deterministically based on the stored seed whenever we sign. This adds a small overhead, while saving a significant amount of storage space. For Dilithium5, we reduce the private key size from 4864 kB to 32 kB. From our benchmarks in Section 6.2, we can see that the speed overhead is 8.2%.

We want to note that discarding information from the secret key during computation is still useful: we need less stack memory to recompute the vector of polynomials t_0 than to store its encoding.

5.3 CTAP Implementation

We added a new algorithm identifier to indicate support for Hybrid between ECDSA and Dilithium. When a Relying Party requests a Hybrid credential, we follow the CTAP procedure as usual. For simplicity, the only change of the encoding of public keys compared to that in ECDSA is the addition of an extra field with the bytes of the Dilithium public key. For registration, we wrap Hybrid credentials as described in Table 2.

Table 2: The private keys and Relying Party ID hash are encrypted with AES-256-CBC. An HMAC-SHA256 of the encrypted data together with the version number is appended.

Bytes	Data description
1	Version number
16	Initialization vector for AES-256
32	ECDSA private key as part of the hybrid
32	Seed for Dilithium private key as part of the hybrid
32	Relying Party ID hashed with SHA256
32	HMAC-SHA256 over everything else

During authentication, the signature is computed with $H(\text{ECDSA}, \text{Dilithium})$. The partial ECDSA signature is ASN.1 DER encoded like standard ECDSA signatures in CTAP.

5.4 Side-channel resilience

As per our attacker model, local attackers are out of scope, and we consider time-based remote side-channels only. Our Dilithium implementation should not leak information about its secret key through the computation time as measurable from outside the device.

The paper introducing Dilithium (Section 5.4 in [17]) explains that their implementation does not leak information about the secret key. Indeed, as Dilithium's signing algorithm may attempt to generate multiple signatures until one that satisfies a set of conditions is found, an adversary can gain information about the number of attempts, or about the conditions previous attempts did not meet. The reasons why a signature attempt is rejected do not depend on the secret key, instead they are based on pseudorandom information. Hence determining which conditions where the reason why a signature attempt was rejected does not help the adversary derive information about the secret key.

Our modifications to Dilithium indeed change the computation compared to the reference implementation. However, our implementation still does not branch depending on the secret data, and hence we maintain the same guarantees.

6 Experiments

We benchmark Dilithium on different target architectures, compare the 3 modes, and evaluate the speed difference of the stack optimized version.

6.1 Dilithium Reference Implementation

Achieving higher security levels demands a higher run time and space usage. Table 3 states the average speed of the Dilithium key generation and signing algorithms over 1000 executions on an x86-64 architecture[4], and the size of the keys and the signature.

Table 3: Average run times on an x86-64 architecture and the key and signature sizes of the reference implementation of Dilithium.

Scheme	Avg runtime of KeyGen (in ms)	Avg runtime of Sign (in ms)	Size of secret key (in bytes)	Size of public key (in bytes)	Size of signature (in bytes)
Dilithium2	0.08	0.31	2528	1312	2420
Dilithium3	0.15	0.53	4000	1952	3293
Dilithium5	0.22	0.61	4864	2592	4595

6.2 Dilithium Embedded

The changes from Section 5.2 enabled us to execute Dilithium in all modes on the Nordic nRF52840 development kit [36]. The performance was measured on the device and the elapsed time printed out via the debugging interface. Table 4 shows the performance we have obtained. In what we call speed mode, we only apply certain stack optimizations to Dilithium, but do not perform the expensive recomputations. This selection allows us to sign messages with Dilithium2, and evaluate the impact of the recomputations directly in this mode. To measure the computational cost of our hybrid scheme, we ran the equivalent experiment to the Dilithium benchmarks, but we use the full hybrid scheme.

If not stated otherwise, all binaries are optimized for size. To compare our runtime to other benchmarks, we also show results compiled for speed in table 5. Note that the code runs as an application on top of an operating system. Therefore, performance benchmarks don't directly compare to other implementations, as some time is spent inside the i.e. syscalls. For an estimate of our relative performance when compiled for speed, we convert the measured time to clock cycles by multiple with the processor speed of 32768 kHz. Those numbers are reported with their relative performance compared to Bos et al. [8].

The binary size of an application running Dilithium on TockOS is 9.3 kB with compiler optimization level -Oz. This size increases to 26.8 kB using -O3.

We highlight that our Dilithium implementation runs solely on the stack; no heap is required. This benefits embedded devices that don't support heap allocation. The memory footprint was measured with stack painting: Before entering the function that we want to measure, we write a fixed byte pattern into the unused stack. After the function returns, we read back the stack to see where the byte pattern was overwritten.

With this method, we can measure the actual stack usage of each function. Therefore, our reported numbers represent our implementation and depend e.g. on the compiler version used. This explains why our numbers are higher than reported theoretical optima

[4]We have used a MacPook Pro (13-inch, 2020), with processor 2.3 GHz Quad-Core Intel Core i7, and memory 16 GB 3733 MHz LPDDR4X.

Table 4: We show the performance obtained by our Optimized Stack mode implementation of Dilithium on the Nordic nRF52840 development kit [36]. The runtime in milliseconds is averaged over 1000 executions, and the stack usage is measured with stack painting. We added runtime speed for ECDSA as a baseline, and to explain the difference between pure Dilithium and Hybrid measurements.

Scheme	Stack usage in kB of		Runtime (in ms) of		Hybrid runtime of	
	KeyGen	Sign	KeyGen	Sign	KeyGen	Sign
ECDSA	0.3	3.0	115.7	188.0		
Dilithium2 (speed mode)	41.6	77.1	70.3	420.4	192.0	687.8
Dilithium2	14.4	17.0	82.3	1053.1	207.5	1417.5
Dilithium3	19.4	17.9	142.4	2077.3	258.5	2420.7
Dilithium5	21.4	19.2	271.4	3305.1	393.1	3378.5

Table 5: We repeated the Dilithium benchmarks from Table 4 with the compiler optimizing for speed rather than binary size (-O3 instead of -Oz) to compare them. We also compare the speed to Bos et al [8] by multiplying our runtimes with our clock frequency.

Scheme	Relative to -Oz		Runtime (in ms) of		Relative to [8]	
	KeyGen	Sign	KeyGen	Sign	KeyGen	Sign
ECDSA	0.45	0.39	51.9	73.3		
Dilithium2 (speed mode)	0.90	0.86	63.2	363.0	0.71	0.64
Dilithium2	0.88	0.91	72.3	956.1	0.81	1.70
Dilithium3	0.91	0.94	129.5	1955.0	0.83	1.76
Dilithium5	0.82	0.82	223.6	2723.8	0.85	2.01

(see [8]). The stack usage is deterministic and does not depend on the inputs' concrete values. Our measurement method also implies that input messages for signing and the RNG are not counted for its memory usage, but outputs are.

Figure 3 summarizes how Dilithium modes scale, and how our stack optimizations impact the speed of operations.

Since Dilithium's signing has a retry loop, its signing speed has a long tail. The distribution of measurements for our Dilithium5 signing benchmark is shown in Figure 4. To not cause timeouts, CTAP operations should be faster than 10 seconds. Signing with Dilithium5 achieves that in 97% of the operations. Key generation is faster and more predictable, taking 271 ms on average, with 1 ms standard deviation.

6.3 Register and Authenticate Speed

Different from pure cryptography measurements above, the performance measurements for the CTAP commands MakeCredential and GetAssertion were measured on the USB host, and include a full message exchange. All measurements use server-side keys (see Section 5.1). MakeCredential takes 792 ms with 2ms standard deviation. GetAssertion has the same long-tail timing distribution as signing (see Figure 4).

We simulated 2000 calls to the security key to register and login. MakeCredential calls took between 786 and 797 milliseconds, whereas GetAssertion has much more variance,

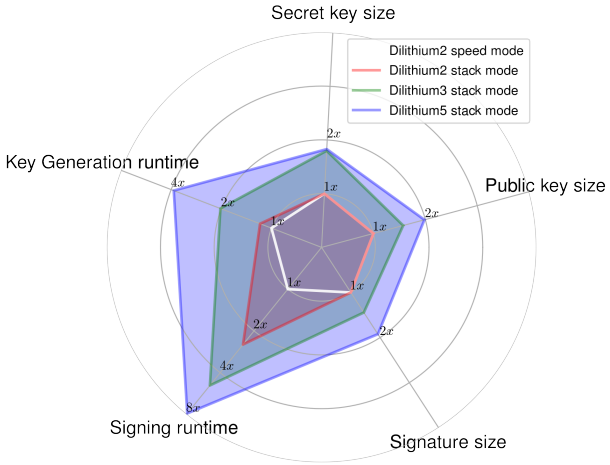


Figure 3: Comparison of sizes and speeds of Dilithium modes on embedded hardware. The reference in white is Dilithium2 without recomputating parts of the key to save memory. To set the computation speed into perspective, we compare the scaling with the key and signatures sizes. Note that the shown key sizes are after restoring from the 32 byte seed.

due to its signing retry logic described above (see **??**). The time distribution shows that 20% of all calls finish within 2 seconds. On average, a command takes 3.9 seconds to complete. 97% of all authentication attempts finished within the CTAP timeout of 10 seconds, as stated in our requirements in Section 5.1.

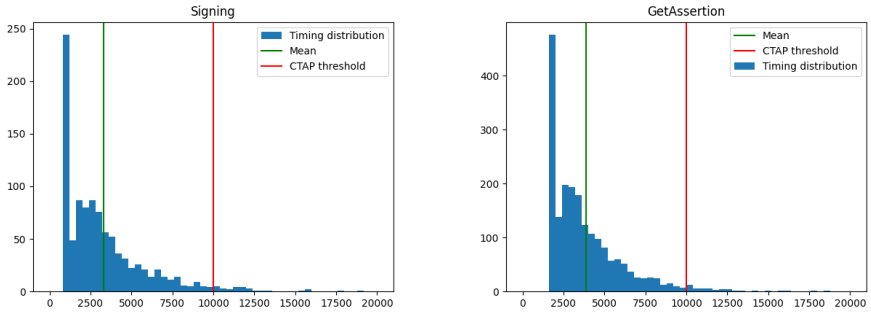
7 Conclusion

In this paper, we proposed a practical way to upgrade security-key authentication via FIDO's CTAP to PQC. To do so, we have designed and evaluated a hybrid digital-signature scheme that combines a classical scheme, ECDSA, with a PQC one, Dilithium. Incidentally, US NIST has recently selected Dilithium as their PQC recommendation for digital signatures. This hybrid scheme ensures that the security guarantees of each underlying scheme are maintained even when one of the scheme becomes insecure.

Our hybrid scheme provides an ulterior layer of security against downgrade attacks, as generated hybrid signatures cannot be used individually. If one of the underlying schemes becomes insecure, this ensures that an attacker cannot leverage the weaker scheme as stand-alone CTAP signature.

To demonstrate the practicality of this scheme, we have implemented it in the open-source security-key firmware OpenSK, benchmarked its performance, and released our contribution as open-source software with an Apache2 license. This way, we encourage other researchers to reproduce our results on a nRF52840 development kit.

Our implementation is designed to overcome the intrinsic resource limitations of current security key hardware platforms while maintaining reasonable run-times. Our evaluation of this implementation has demonstrated its feasibility even when using Dilithium's highest security mode, which comes with the highest resource requirements.



(a) The sign operation has a retry loop that discards insecure parameters. The signing speed is therefore highly non-deterministic.

(b) GetAssertion commands have a similar long tail, depending on the number of retries when signing with Dilithium.

Figure 4: Timing distributions of signing and the CTAP command GetAssertion, using the Dilithium5 mode.

8 Appendix

8.1 Proofs

We include the formal proofs that were omitted in the main body of the paper.

Lemma 2. *If Σ_1 is X-EUF-CMA secure, then $H(\Sigma_1, \Sigma_2)$ is X-EUF-CMA secure as well.*

Proof. For every X-adversary A that wins the X-EUF-CMA security game for $H(\Sigma_1, \Sigma_2)$ with probability p_A, there is an adversary B that wins Σ_1's X-EUF-CMA security game with probability $p_B \geq p_A$. B can be constructed as follows:

- B receives the Σ_1 public key pk_1 from the challenger C_{Σ_1}. B generates its own Σ_2 keys $(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow_\$ \Sigma_2.\mathtt{KeyGen}(1^\kappa)$ and sends $\mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2)$ to A. Note that the keys received by A are generated from the same probability distribution as in the X-EUF-CMA security game of $H(\Sigma_1, \Sigma_2)$.
- When receiving a message query m_i from A, B sends its message query $m_i := \mathtt{label}\ m_i$ to C_{Σ_1} and obtains $\sigma_{1,i} = \Sigma_1.\mathtt{Sign}(m_i, \mathsf{sk}_1)$. Then, B uses its own Σ_2 secret key to compute $\sigma_{2,i} \leftarrow_\$ \Sigma_2.\mathtt{Sign}((m_i, \sigma_{1,i}), \mathsf{sk}_2)$ and sends $\sigma_i := (\sigma_{1,i}, \sigma_{2,i})$ to A. Note that the hybrid signature is valid: $H(\Sigma_1, \Sigma_2).\mathtt{Verify}(m_i, \sigma_i, \mathsf{pk}) = \mathsf{true}$.
- When receiving the forgery $(m\ , \sigma\ = (\sigma_1, \sigma_2))$ from A, B obtains its own forgery $(\mathtt{label}\ m\ , \sigma_1)$ and sends it to C_{Σ_1}.

Since B simulates the X-EUF-CMA security game for $H(\Sigma_1, \Sigma_2)$ perfectly towards A, A maintains its success probability p_A. Additionally, whenever A wins the simulated game, B wins the X-EUF-CMA security game for Σ_1: A wins if $m\ \ / \{\text{queries } m_i\}$ and $H^{SN}.\mathtt{Verify}(m\ , \sigma\ , \mathsf{pk}) = \mathsf{true}$, which implies that $\mathtt{label}\ m\ \ / \{\text{queries } m_i\}$ and $\Sigma_1.\mathtt{Verify}(\mathtt{label}\ m\ , \sigma_1, \mathsf{pk}_1) = \mathsf{true}$. Hence, B wins Σ_1's X-EUF-CMA security game with probability $p_B \geq p_A$.

Finally, as Σ_1 is X-EUF-CMA secure, $p_B\ \ \mathsf{negl}(\kappa)$, and therefore $p_A\ \ \mathsf{negl}(\kappa)$. Since A was chosen arbitrarily, we obtain that any X-adversary has negligible probability in winning $H(\Sigma_1, \Sigma_2)$'s X-EUF-CMA security game. □

Lemma 3. *If Σ_2 is X-EUF-CMA secure (resp. X-SUF-CMA) secure, $H(\Sigma_1, \Sigma_2)$ is X-EUF-CMA secure (resp. X-SUF-CMA) as well.*

Proof. For every X-adversary $\mathcal{A}$ that wins the X-EUF-CMA (resp. X-SUF-CMA) security game for $\mathcal{H}(\Sigma_1, \Sigma_2)$ with probability $p_{\mathcal{A}}$, there is an adversary $\mathcal{B}$ that wins Σ_2's X-EUF-CMA (resp. X-SUF-CMA) security game with probability $p_{\mathcal{B}} \geq p_{\mathcal{A}}$. $\mathcal{B}$ can be constructed as follows:

- $\mathcal{B}$ receives the Σ_2 public key pk_2 from the challenger $\mathcal{C}_{\Sigma_2}$. $\mathcal{B}$ generates its own pair of Σ_1 keys $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow_\$ \Sigma_1.\mathsf{KeyGen}(1^\kappa)$ and sends $\mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2)$ to $\mathcal{A}$. Note that the keys received by $\mathcal{A}$ are generated from the same probability distribution as in $\mathcal{H}(\Sigma_1, \Sigma_2)$'s security game.
- When receiving a message query m_i from $\mathcal{A}$, $\mathcal{B}$ uses its own secret key sk_1 to compute $\sigma_{1,i} \leftarrow_\$ \Sigma_1.\mathsf{Sign}(\mathtt{label}\ m_i, \mathsf{sk}_1)$. Then, $\mathcal{B}$ sends its message query $m_i := (\mathtt{label}\ m_i, \sigma_1)$ to $\mathcal{C}_{\Sigma_2}$ and obtains $\sigma_{2,i} = \Sigma_2.\mathsf{Sign}(m_i, \mathsf{sk}_2)$. Finally, $\mathcal{B}$ sends the hybrid signature $\sigma_i := (\sigma_{1,i}, \sigma_{2,i})$ to $\mathcal{A}$. Note that σ_i is a valid hybrid signature: $\mathcal{H}(\Sigma_1, \Sigma_2).\mathsf{Verify}(m_i, \sigma_i, \mathsf{pk}) = \mathsf{true}$.
- When receiving the forgery $\big(m\ , \sigma\ = (\sigma_1, \sigma_2)\big)$ from $\mathcal{A}$, $\mathcal{B}$ obtains its own forgery $\big((\mathtt{label}\ m\ , \sigma_1), \sigma_2\big)$ and sends it to $\mathcal{C}_{\Sigma_2}$.

Since $\mathcal{B}$ simulates the X-EUF-CMA (resp. X-SUF-CMA) security game for $\mathcal{H}$ perfectly towards $\mathcal{A}$, $\mathcal{A}$ maintains its success probability $p_{\mathcal{A}}$. In addition, whenever $\mathcal{A}$ wins the simulated game, $\mathcal{B}$ wins the X-EUF-CMA (resp. X-SUF-CMA) security game for Σ_2. If $\mathcal{A}$ wins the simulated X-EUF-CMA security game, then $m\ \ / \{\text{queries } m_i\}$. It immediately follows that $\mathtt{label}\ m\ \ / \{\text{queries } m_i\}$. If $\mathcal{A}$ wins the simulated X-SUF-CMA security game, then $(m\ , \sigma\ )\ / \{(m_i, \sigma_i) \mid m_i \text{ query}\}$. If this is the case, if $\big((\mathtt{label}\ m\ , \sigma_1), \sigma_2\big) = (m_i, \sigma_{2,i})$ for some query m_i, we obtain that $(m\ , \sigma\ ) = (m_i, \sigma_i)$, which contradicts that $\mathcal{A}$'s forgery was successful. In both of the cases, $\mathcal{H}(\Sigma_1, \Sigma_2).\mathsf{Verify}(m\ , \sigma\ , \mathsf{pk}) = \mathsf{true}$ must hold, hence $\Sigma_2.\mathsf{Verify}((\mathtt{label}\ m\ , \sigma_1), \sigma_2, \mathsf{pk}_2)$ holds as well. It follows that that $\mathcal{B}$ wins the X-EUF-CMA (resp. X-SUF-CMA) security game for Σ_2 with probability $p_{\mathcal{B}} \geq p_{\mathcal{A}}$.

Finally, as Σ_2 is X-EUF-CMA (resp. X-SUF-CMA) secure, $p_{\mathcal{B}}\ \ \mathsf{negl}(\kappa)$, and therefore $p_{\mathcal{A}}\ \ \mathsf{negl}(\kappa)$. Since $\mathcal{A}$ was chosen arbitrarily, we obtain that any X-adversary has negligible probability in winning $\mathcal{H}(\Sigma_1, \Sigma_2)$'s X-EUF-CMA (resp. X-SUF-CMA) security game. □

Lemma 4. *Let $\mathcal{U}_{\mathtt{label}}$ denote the set $\{m \mid m$ does not have prefix $\mathtt{label}\}$. If Σ_1 is X-EUF-CMA secure, $\mathcal{H}(\Sigma_1, \Sigma_2)$ is $(1, X, \mathcal{U}_{\mathtt{label}})$-non-separable.*

Proof. For every X-adversary $\mathcal{A}$ that wins the $(1, X, \mathcal{U}_{\mathtt{label}})$-non-separability security game for $\mathcal{H}(\Sigma_1, \Sigma_2)$ with probability $p_{\mathcal{A}}$, there is an adversary $\mathcal{B}$ that wins Σ_1's X-EUF-CMA security game with probability $p_{\mathcal{B}} \geq p_{\mathcal{A}}$. $\mathcal{B}$ can be constructed as follows:

- $\mathcal{B}$ receives the Σ_1 public key pk_1 from the challenger $\mathcal{C}_{\Sigma_1}$. $\mathcal{B}$ generates its own pair of Σ_2 keys $(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow_\$ \Sigma_2.\mathsf{KeyGen}(\kappa)$ and sends $\big(\mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2), \{\mathsf{pk}_1\}\big)$ to $\mathcal{A}$. Note that the public keys received by $\mathcal{A}$ are generated from the same probability distribution as in $\mathcal{H}(\Sigma_1, \Sigma_2)$'s $(1, X, \mathcal{U}_{\mathtt{label}})$-non-separability game.
- When receiving a message query m_i from $\mathcal{A}$, $\mathcal{B}$ sends its message query $m_i := \mathtt{label}\ m_i$ to $\mathcal{C}_{\Sigma_1}$ and obtains $\sigma_{1,i} = \Sigma_1.\mathsf{Sign}(m_i, \mathsf{sk}_1)$. Then, $\mathcal{B}$ uses its own Σ_2 secret key to compute $\sigma_{2,i} \leftarrow_\$ \Sigma_2.\mathsf{Sign}((m_i, \sigma_{1,i}), \mathsf{sk}_2)$ and sends $\big(\sigma_i := (\sigma_{1,i}, \sigma_{2,i}),\ \big)$ to $\mathcal{A}$. Note that the hybrid signature σ_i is valid: $\mathcal{H}(\Sigma_1, \Sigma_2).\mathsf{Verify}(m_i, \sigma, \mathsf{pk}) = \mathsf{true}$.
- When receiving the forgery $(m\ , \sigma\ , \mathsf{pk}\ )$ from $\mathcal{A}$, $\mathcal{B}$ sends its forgery $(m\ , \sigma\ )$ to $\mathcal{C}_{\Sigma_1}$.

As $\mathcal{B}$ simulates the $(1, X, \mathcal{U}_{\mathtt{label}})$-non-separability game for $\mathcal{H}(\Sigma_1, \Sigma_2)$ perfectly towards $\mathcal{A}$, $\mathcal{A}$ maintains its success probability $p_{\mathcal{A}}$ in the simulated game.

In addition, whenever $\mathcal{A}$ wins the simulated game, $\mathcal{B}$ wins Σ_1's X-EUF-CMA security game: $\mathcal{A}$ wins if $m\ \ \mathcal{U}_{\mathtt{label}}$, $\mathsf{pk}\ \ \{\mathsf{pk}_1\}$, and $\Sigma_1.\mathsf{Verify}(m\ , \sigma_1, \mathsf{pk}\ ) = \mathsf{true}$. If this is

the case, as each query m_i sent by B to C_{Σ_1} has the prefix `label`, $m \not\in \{\text{queries } m_i\}$. Hence, B outputs a successful forgery with success probability is $p_B \geq p_A$.

Finally, as Σ_1 is X-EUF-CMA secure, $p_B \leq \mathsf{negl}(\kappa)$, and therefore $p_A \leq \mathsf{negl}(\kappa)$. Since A was chosen arbitrarily, we obtain that any X-adversary has negligible probability in winning $H(\Sigma_1, \Sigma_2)$'s $(1, X, U_{\texttt{label}})$-non-separability security game. □

We omit the formal proof of Lemma 5 as it follows the same rationale as the proof of Lemma 4.

References

[1] Amin Abdulrahman, Vincent Hwang, Matthias J. Kannwischer, and Daan Sprenkels. Faster Kyber and Dilithium on the Cortex-M4. In *ACNS*, volume 13269 of *Lecture Notes in Computer Science*, pages 853–871. Springer, 2022.

[2] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[3] Reza Azarderakhsh, Rami Elkhatib, Brian Koziel, and Brandon Langenberg. Hardware Deployment of Hybrid PQC: SIKE+ECDH. In *International Conference on Security and Privacy in Communication Systems*, pages 475–491. Springer, 2021.

[4] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical Stateless Hash-Based Signatures. *IACR Cryptology ePrint Archive*, 2014:795, 2014.

[5] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. Leveraging Secondary Storage to Simulate Deep 54-qubit Sycamore Circuits. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 2129–2146, 2019.

[6] Ward Beullens. Improved Cryptanalysis of UOV and Rainbow. In *Advances in Cryptology – EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I*, page 348–373, Berlin, Heidelberg, 2021. Springer-Verlag.

[7] Nina Bindel, Udyani Herath, Matthew McKague, and Douglas Stebila. Transitioning to a Quantum-Resistant Public Key Infrastructure. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography*, pages 384–405, Cham, 2017. Springer International Publishing.

[8] Joppe W Bos, Joost Renes, and Daan Sprenkels. Dilithium for memory constrained devices. *Cryptology ePrint Archive*, 2022.

[9] Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. The Provable Security of Ed25519: Theory and Practice. Cryptology ePrint Archive, Paper 2020/823, 2020. https://eprint.iacr.org/2020/823.

[10] D. Brown. *On the Provable Security of ECDSA*, page 21–40. London Mathematical Society Lecture Note Series. Cambridge University Press, 2005.

[11] Daniel Brown. Generic groups, Collision Resistance, and ECDSA. *Design, Codes Cryptography*, 35, 03 2002.

[12] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS-A Practical Forward Secure Signature Scheme based on Minimal Security Assumptions. In *International Workshop on Post-Quantum Cryptography*, pages 117–129. Springer, 2011.

[13] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. Report on Post-Quantum Cryptography. Technical report, National Institute of Standards and Technology, April 2016.

[14] CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation. https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf. Accessed: 2022-07-08.

[15] Client to Authenticator Protocol (CTAP). https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-20210615.html. Accessed: 2022-04-05.

[16] Dowling, Benjamin, Brandt Hansen, Torben, and Paterson, Kenneth G. Many a Mickle Makes a Muckle: A Framework for Provably Quantum-Secure Hybrid Key Exchange. *Lecture Notes in Computer Science*, 2020.

[17] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):238–268, Feb. 2018.

[18] Shimon Even, Oded Goldreich, and Silvio Micali. On-Line/Off-Line Digital Signatures. In *Advances in Cryptology (CRYPTO'89)*, volume 435, pages 263–275, 08 1989.

[19] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the Provable Security of (EC)DSA Signatures. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1651–1662, New York, NY, USA, 2016. Association for Computing Machinery.

[20] FIDO Alliance. https://fidoalliance.org/. Accessed: 2022-04-05.

[21] FIDO Alliance security reference. https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-security-ref-v2.0-id-20180227.html. Accessed: 2022-04-05.

[22] Denisa OC Greconici, Matthias J Kannwischer, and Daan Sprenkels. Compact Dilithium Implementations on Cortex-M3 and Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 1–24, 2021.

[23] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking nist pqc on arm cortex-m4. Cryptology ePrint Archive, Paper 2019/844, 2019. https://eprint.iacr.org/2019/844.

[24] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A Concrete Treatment of Fiat-Shamir Signatures in the Quantum Random-Oracle Model. In *Advances in Cryptology – EUROCRYPT 2018*, pages 552–586. Springer International Publishing, 2018.

[25] Neal Koblitz. Mathematics of Computation. *Elliptic Curve Cryptosystems, 48n*, 1(77):l, 1987.

[26] Information Technology Laboratory. Digital Signature Standard (DSS). Technical report, National Institute of Standards and Technology, July 2013.

[27] Juan Lang, Alexei Czeskis, Dirk Balfanz, and Marius Schilder. Security Keys: Practical Cryptographic Second Factors for the Modern Web. In *Financial Cryptography*, 2016.

[28] Frank T Leighton and Silvio Micali. Large provably fast and secure digital signature schemes based on secure hash functions, July 11 1995. US Patent 5,432,852.

[29] Amit Levy, Bradford Campbell, Branden Ghena, Daniel B. Giffin, Pat Pannuto, Prabal Dutta, and Philip Levis. Multiprogramming a 64kB Computer Safely and Efficiently. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP'17, pages 234–251, New York, NY, USA, 10 2017. ACM.

[30] Shaohua Li, Kaiping Xue, Chenkai Ding, Xindi Gao, David S. L. Wei, Tao Wan, and Feng Wu. FALCON: A Fourier Transform Based Approach for Fast and Secure Convolutional Neural Network Predictions. *CoRR*, abs/1811.08257, 2018.

[31] Benjamin Lipp. An Analysis of Hybrid Public Key Encryption. Cryptology ePrint Archive, Paper 2020/243, 2020. https://eprint.iacr.org/2020/243.

[32] Soundes Marzougui, Vincent Ulitzsch, Mehdi Tibouchi, and Jean-Pierre Seifert. Profiling Side-Channel Attacks on Dilithium: A Small Bit-Fiddling Leak Breaks It All. Cryptology ePrint Archive, Paper 2022/106, 2022. https://eprint.iacr.org/2022/106.

[33] Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque. Masking Dilithium. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 344–362, Cham, 2019. Springer International Publishing.

[34] NIST Announces First Four Quantum-Resistant Cryptographic Algorithms. https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms. Accessed: 2022-07-07.

[35] NIST Post-Quantum Cryptography FAQs. https://csrc.nist.gov/Projects/post-quantum-cryptography/faqs. Accessed: 2022-07-13.

[36] Nordic nrf52840. https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dk. Accessed: 2022-04-05.

[37] OpenSK. https://github.com/google/OpenSK. Accessed: 2022-04-05.

[38] PQCrystals: Dilithium. https://github.com/pq-crystals/dilithium. Accessed: 2022-06-10.

[39] John Proos and Christof Zalka. Shor's Discrete Logarithm Quantum Algorithm for Elliptic Curves. *arXiv preprint quant-ph/0301141*, 2003.

[40] Manohar Raavi, Simeon Wuthier, Pranav Chandramouli, Yaroslav Balytskyi, Xiaobo Zhou, and Sang-Yoon Chang. Security Comparisons and Performance Analyses of Post-quantum Signature Algorithms. In *International Conference on Applied Cryptography and Network Security*, pages 424–447. Springer, 2021.

[41] Prasanna Ravi, Sourav Sen Gupta, Anupam Chattopadhyay, and Shivam Bhasin. Improving Speed of Dilithium's Signing Procedure. In *International Conference on Smart Card Research and Advanced Applications*, pages 57–73. Springer, 2019.

[42] Ronald L Rivest, Adi Shamir, and Leonard M Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. In *Secure communications and asymmetric cryptosystems*, pages 217–239. Routledge, 2019.

[43] Peter W Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM review*, 41(2):303–332, 1999.

[44] Teik Guan Tan, Pawel Szalachowski, and Jianying Zhou. Challenges of Post-Quantum Digital Signing in Real-world Applications: A Survey. *International Journal of Information Security*, pages 1–16, 2022.