Post-Quantum Secure Boot on Vehicle Network Processors

Joppe W. Bos*, Brian Carlson*, Joost Renes*, Marius Rotaru*, Daan Sprenkels†, and Geoffrey P. Waters*

NXP Semiconductors
* firstname.lastname@nxp.com
† daan@dsprenkels.com

Abstract. The ability to trust a system to act safely and securely strongly relies on the integrity of the software that it runs. To guarantee authenticity of the software one can include cryptographic data such as digital signatures on application images that can only be generated by trusted parties. These are typically based on cryptographic primitives such as Rivest-Shamir-Adleman (RSA) or Elliptic-Curve Cryptography (ECC), whose security will be lost whenever a large enough quantum computer is built. For that reason, migration towards Post-Quantum Cryptography (PQC) is necessary. This paper investigates the practical impact of migrating the secure boot flow on a Vehicle Network Processor (S32G274A) towards PQC. We create a low-memory fault-attack-resistant implementation of the Dilithium signature verification algorithm and evaluate its impact on the boot flow.

Keywords: Post-Quantum Cryptography · Digital Signatures · Secure Boot · Automotive Processors · S32G274A

1 Introduction

The concept of digital signatures, invented by Diffie and Hellman [6], is one of the core cryptographic building blocks to construct secure protocols. Its main goal is to provide message authentication against (the public key of) a sender. Given a signature with respect to a message and public key, it should be impossible (i.e., computationally infeasible) to modify either message or public key without verification of the signature failing. Typical use cases include verifying a signature over a software update, or over firmware installed on a device during the boot process, to prevent any malicious modification of the intended image. These serve as the basis of trust for any other applications running on a system. For example, modern cars feature service-oriented gateways that are responsible for transfering data between various vehicle networks, handle Over-The-Air (OTA) updates, communicate with the cloud, etc.

Secure Boot. The goal of secure boot is to guarantee integrity and authenticity of the software running on a system. Although there are different ways in which to achieve this, ultimately the confidence in a system leads back to a so-called *Root of Trust* (RoT). For example, an RoT can consist of executable code and (hashes of) keys in Read-Only Memory (ROM) that performs various initializations, verifies the authenticity of the firmware, and finally passes control to the (now authenticated) firmware. The requirements on RoTs are well-documented by various organizations, e.g., see TCG [4, Part 1 §9.5.5] or GlobalPlatform [10], and its implementation should hold up against strong testing and certification (e.g., ISO 26262) requirements. In particular, in order to prevent any of the ROM code being modified, or executable instructions skipped altogether, the RoT should be protected against physical fault attacks [3,2].

Since both security requirements as well as cost of implementation for RoTs are high, their design typically aims to provide the necessary security requirements with minimal footprint. As such, in most modern systems the boot flow is not completed when the ROM code passes on control. Instead, more advanced features are offloaded to a (second-stage) *boot loader*, which is verified by ROM code and made responsible for the remainder of the boot sequence. Of course, this boot loader can in turn verify and advance control to the next stage, creating a *chain of trust*. In complex ecosystems distinct parties can be responsible for the different stages in the chain: while immutable hardware such as ROM needs to be established at manufacturing time by a Tier-1 or Tier-2 supplier, second (or higher) stage boot loaders can rely on memory that is programmed only later in the process by Original Equipment Manufacturers (OEMs), for example. This chain can extend all the way to end users running their Operating System (OS) of choice.

Post-Quantum Digital Signatures. All widely used and deployed approaches to realize digital signatures are either based on Elliptic Curve Cryptography (ECC) [21,29] or the Rivest–Shamir–Adleman (RSA) [36] algorithm. However, with the steady progress in the development of a quantum computer, the security of these approaches is threatened. If large-scale quantum computers are to become a reality, Shor's algorithm [40] will be able to recover ECC/RSA private keys in polynomial time. To prepare for this threat, alternative public-key algorithms are necessary. These are typically referred to as post-quantum, or quantum-safe, algorithms.

As opposed to the already well-established state of classical digital signatures, post-quantum alternatives are still very much in development. One promising direction is that of *hash-based* signatures, which is arguably the most mature variant. The two instantiations eXtended Merkle Signature Scheme (XMSS) and Leighton-Micali Signatures [28] (LMS) have been established as Requests For Comments (RFCs) by the Internet Engineering Task Force (IETF) [15,28], and have recently been published as a NIST Special Publication 800-208 [5]. They are based on well-established cryptographic primitives (i.e., hash functions) and have very fast signature verification times. The main downside is that both signa-

ture schemes are *stateful* on the end of the signer, which can seriously complicate key management. Their *stateless* counterpart SPHINCS+ [14] solves this issue, but at the cost of significantly increasing the signature size. Other alternatives explore digital signatures based on the hardness of multivariate quadratic equations, most notably the Rainbow scheme [7], or based on error-correcting codes and isogenies. Unfortunately either their keys or signatures are extremely large, or they are relatively inefficient, making them difficult to apply in embedded scenarios.

We focus instead on *lattice-based* digital signatures, which have a rich history and a wide variety of options relying on different (but related) hardness assumptions. One approach in lattice-based cryptography is based on Regev's work introducing the Learning With Errors (LWE) problem [35], which relates to solving a "noisy" linear system modulo a known integer. This problem can be used as the basis for a signature scheme, as shown by Lyubashevsky [25], by improving on his idea to apply Fiat-Shamir with aborts [24] to lattices. A more specialized version is based on the Ring Learning With Errors (R-LWE) problem [27,32], which works in a special ring (more specifically the ring of integers of a cyclotomic number field) that offers significant storage and efficiency improvements compared to LWE. Although R-LWE has additional algebraic structure and relies on the (worst-case) hardness of problems in *ideal* lattices, no significant concrete improvements in cryptanalysis are known. Finally, a combination of many of these ideas (plus various improvements) resulted in CRYSTALS–Dilithium [8] based on *Module*-LWE.

In an effort to standardize such algorithms the US National Institute of Standards and Technology (NIST) put out a call for proposals [30] to submit candidate algorithms in 2016. As of July 2020, seven out of fifteen remaining candidates have been marked as finalists of which a subset is expected to be standardized in 2022.

Related work. Sanwald, Kaneti, Stöttinger and Böhner performed a thorough investigation of secure boot in the automotive domain [37]. Integration of post-quantum secure key exchange and digital signature verification has been studied before. The main investigations have been around hash-based signature schemes, since they have already been standardized by NIST [5]. They come with some potential disadvantages of requiring to keep a state during signature generation. An impact assessment of hash-based post-quantum secure schemes on secure boot is studied by Kampanakis, Panburana, Curcio and Shroff [19]. Hermelink, Pöppelmann, Stöttinger, Wang and Wan perform an investigation into Authenticated Key Exchange (AKE) combining XMSS and NewHope [13], while Feritzmann, Vith, Flórez and Sepúlveda analyze lattice-based Key Encapsulation Mechanisms (KEMs) for automotive systems [9]. Also, Kumar, Gupta, Chattopadhyay, Kasper, Krauß and Niederhagen [23] investigate how hash-based schemes can be integrated into a secure SoC platform around RISC-V cores and evaluated on an FPGA.

As far as we are aware, integration of lattice-based schemes into the secure boot flow has been not investigated before. In this work we focus on the NIST signature finalist Dilithium. Dilithium is often considered for embedded applications due to its favorable runtime and relatively small size, for example the embedded implementation from [12,34] and the improvements presented in [11].

Our contributions. We investigate the practical impact of protecting the secure boot flow for vehicle network processors against quantum attacks. This is realized by integrating the Dilithium digital signature scheme into the secure boot process of the S32G platform. As part of this work we created a fault-attack resistant (against single-targeted faults) Dilithium signature verification algorithm, which uses significantly less memory than the state of the art. This implementation was integrated in the S32G Hardware Security Engine (HSE) secure boot flow (cf. Section 4). We have measured the latency of our Dilithium verification algorithm in a regular setting and using *pre-hashing* with SHA-256. Our results (Section 4.3) make it clear that the use of post-quantum cryptography does have an impact on the (one-time) installation time of an application image. However after installation, the S32G uses a reference proof instead of verifying the original signature, which means that the boot time is not affected by the signature scheme. We evaluate these results and find that the impact is fairly minimal: a transition to post-quantum secure boot can be considered practical for this application.

Organization. We begin with a description of the Dilithium digital signature scheme in Section 2. We describe the S32G and its (secure) boot flow in Section 3, and the integration results of Dilithium in Section 4. Finally, we present our conclusions in Section 5.

2 Preliminaries

The goal of this section is to provide some background information on the Dilithium digital signature scheme, For a full description we refer to its specification [8,26].

The security of Dilithium is based on hard problems related to *ideal* lattices, as opposed to popular classical schemes based on the difficulty of factoring (RSA) or solving discrete logarithms (e.g., DSA, ECDSA, EdDSA). More precisely, it relies on the hardness of Module Learning With Errors (M-LWE) [27], or (variants of) Module Short Integer Solution (M-SIS). Intuitively, the first underlies the key generation where the public part of the key $\mathbf{t}$ is computed as

$$\mathbf{t} = \mathbf{A}\mathbf{s_1} + \mathbf{s_2}\,,$$

where the secrets $\mathbf{s_1}$ and $\mathbf{s_2}$ are generated from a small centered binomial distribution and $\mathbf{A}$ a random $k \times \ell$ matrix (where k and ℓ depend on the security level). Recovering $\mathbf{s_1}$ and/or $\mathbf{s_2}$ from $\mathbf{t}$ is by definition as hard as solving M-LWE.

On the other hand, the M-SIS assumption can be used to show that forging signatures is infeasible.

Although the underlying mathematical structures are very different from their classical counterparts, the design of the signature scheme itself is interestingly enough quite similar. The construction works by first defining an interactive *sigma* protocol, consisting of a commitment, challenge and response. This is turned into a *non-interactive* signature scheme using the Fiat-Shamir heuristic. Analogous to variants of Schnorr identification and signature schemes, the reponse for Dilithium is generated as $\mathbf{z} = \mathbf{y} + c\mathbf{s_1}$. However, crucially, this relation holds over the integers as opposed to a prime field as is the case for Schnorr. This has the particular consequence that the distribution of $\mathbf{z}$ (which is part of the signature) is not necessarily independent of $\mathbf{s_1}$, potentially leaking information about the secret key. It can be shown that $\mathbf{z}$ is independent from $\mathbf{s_1}$ if and only if $\|\mathbf{z}\|_\infty$ is large enough, which is explicitly checked before outputting $\mathbf{z}$. If this fails, the signing procedure is aborted and restarted with an updated counter. This design is called *Fiat-Shamir with aborts* and leads to *variable-time* signature creation [24]. Fortunately, this does not impact the constant-time signature verification.

Dilithium supports signatures on messages M of abritrary length by initiating the signing procedure with $\mu = \texttt{SHA3}(\texttt{SHA3}(pk)\|M)$. That is, a hash over the message prepended with a hash over the public key, binding the message to the public key. However, there are limitations to directly applying this to the message (e.g., a large application image). Firstly, by design of Dilithium this fixes the choice of SHA3 as the message hash algorithm. For large messages the initial hash can be the deciding factor for the efficiency of signing, making efficient implementation critical. However, especially on existing platforms, hardware acceleration for SHA3 is not necessarily available, while it might be for other hash functions. As we shall see in Table 1, for an image of 128 KiB on the S32G274A processor the choice between $\mu = \texttt{SHA3}(\texttt{SHA3}(pk)\|M)$ and $\mu = \texttt{SHA3}(\texttt{SHA3}(pk)\|\texttt{SHA2}(M))$ is the difference between the initial hash causing a huge slowdown or having negligible impact.

The scheme can be instantiated for three security levels as defined by NIST. Its lowest security level (Level 2) is designed such that the computational resources required to break the security are at least as high as those needed to those needed for finding collisions on a 256-bit hash function. Its medium (Level 3) and high (Level 3) security levels are constructed such that they are at least as hard to break as recovering a key for a 192-bit and 256-bit (ideal) block cipher, respectively. The respective signatures consist of 2420, 3293 and 4595 bytes, while the public keys are 1312, 1952 and 2592 bytes long. This is relatively small compared to most other digital signature finalists in the NIST standardization effort. However, when we compare this to the size of RSA and ECC keys and signatures the post-quantum equivalents are still significantly larger compared to what we are used to.

3 S32G Vehicle Network Processors

3.1 Platform Description

In this work we use an *S32G vehicle network processor* as the target platform for the impact assessment of integrating post-quantum cryptography in the secure boot flow. This high-end automotive processor is developed by NXP Semiconductors and part of a larger S32 automotive platform which includes the S32K, S32R and S32V and is designed to meet the safety and security requirements in the automotive and industrial domains (i.e., compliance with IEC 61508 [16] and ASIL-D classification in ISO 26262 [17]). Typical uses include service-oriented gateways, domain controllers, vehicle computers and safety processors. The S32G consists of a combination of microcontrollers (MCUs) based on the Arm Cortex-M7, and microprocessors (MPUs) based on Arm Cortex-A53. These Application CPUs are combined with several types of memory (SRAM, DRAM, NOR/NAND Flash) and various hardware accelerators. Most notably, it contains a Hardware Security Engine (HSE) which supports both symmetric and (classical) asymmetric cryptography accelerators, a random number generator, and dedicated secure memory. The HSE serves both as a secure environment for host applications, as well as being responsible for (part of) the boot flow if secure boot is enabled.

The precise configuration depends on the choice of model: we deploy the S32G274A which contains 3 Arm Cortex-M7 cores, 4 Arm Cortex-A53 cores, and 8 MB of system RAM. Each of the MCUs runs in a delayed lockstep configuration at a maximum frequency of 400 MHz and has 32 KB instruction and data caches. The MPUs are configured as 2 clusters of 2 cores each running at a maximum frequency of 1 GHz. Every core has access to 32 KB L1 instruction and data caches, while each cluster shares another 512 KB of L2 cache. Optionally, the A53 clusters can be configured to also run in a delayed lockstep setting, effectively removing one of the clusters from an application's point of view but increasing the fault tolerance.

3.2 Secure Boot on the S32G274A

The S32G274A provides two modes of startup, a normal-start-up sequence (also referred to as "normal boot") and a secure start-up sequence (also referred to as "secure boot"). The secure boot process involves a series of stages. In each stage, a new piece of code is executed after passing all necessary checks for authenticity and optionally decryption of the protected content. In case the authentication fails, the related CPU subsystem remains in reset, potentially rendering the device inoperant, or at least not operating as originally designed for the targeted application

In the secure startup process, the S32G starts operating a trusted-boot stored Read-Only Memory (ROM) firmware (BootROM), that is responsible for verifying, decrypting, and loading the HSE Security firmware (HSE-FW) into HSE secure memory before handing over the control to it. Once HSE-FW is up and running, it is responsible for initiating the next boot stage, by verifying and

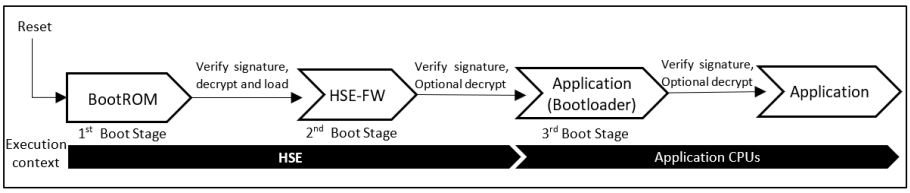

Fig. 1. Secure boot flow for the S32G274A.

optionally decrypting the Application (Bootloader), before starting the Application CPUs. The authenticity check is based on cryptographic primitives. In particular, digital signatures schemes that are supported for authenticating application images are RSA [36] with various padding schemes [33], ECDSA [38], and EdDSA [1,31].

The HSE requires three essential components for the boot sequence, which need to be installed before enabling secure boot. For example, this can be done by executing an application that performs the installation via the normal boot sequence, or through the serial interface. Firstly, any user keys (i.e., those not already in ROM) that are to be used by the HSE-FW to check the authenticity of the application need to be provisioned. Secondly, the application images need to be installed in non-volatile memory. This is done with the use of Secure Memory Regions (SMRs), which are regions in Non-Volatile Memory (NVM) defined by an address, a length, and an (initial) proof of authenticity (e.g., a digital signature linked to a previously provisioned key). Finally, the user has to specify the Application CPUs for which SMRs require verification before continuing with the boot flow (and which sanctions are applied on failure). The Application (Bootloader) and the Application can be associated with one or more SMRs. The HSE secure boot configuration can be locked by advancing the device lifecycle, disabling any future changes to the configuration.

It should be noted that the above description is a very high-level view: in reality, the S32G274A boot sequence is highly configurable and supports a multitude of options. A particularly interesting one is the ability to use reference proofs of authenticity for SMRs. On initial SMR installation, the HSE-FW will check the initial authenticity proof that was stored in non-volatile memory (e.g., the digital signature). If the initial authenticity proof verifies correctly, the HSE-FW computes a reference authenticity proof that is stored internally in the HSE. As the application has already been authenticated with an initial proof of authenticity, the requirements on the reference proof are lighter. Therefore verification of the reference proof can be much faster than the initial one. During secure boot, the HSE-FW only verifies the SMRs by checking the reference proofs, significantly speeding up the boot procedure. The S32G also supports runtime (periodically or on-demand) attestation, meaning that SMRs can be verified (initial or reference) during the execution.

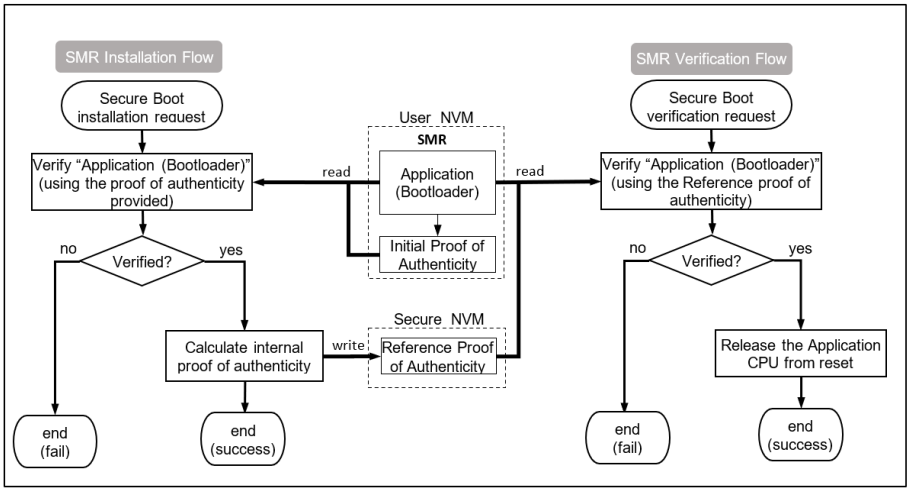

Fig. 2. Overview of booting using Secure Memory Regions.

4 S32G274A Post-Quantum Secure Boot

In Section 3 we summarized the S32G platform and its boot flow. In this section, we describe our Dilithium implementation for the HSE core and how this was integrated into the HSE-FW to support its signature verification in the boot flow. Finally, we discuss the installation of the secure memory regions selecting features that are most appropriate for our setting.

4.1 Dilithium Software

The Dilithium submission to the NIST standardization effort is accompanied by various implementations [26]. Some of the parameter sets for Dilithium have also been integrated and optimized for *pqm4*: a testing and benchmarking framework for the Arm Cortex-M4 [20]. The implementations supported in pqm4 provide a good overview of the state-of-the-art performance of the post-quantum algorithms within some constraints related to the Arm Cortex-M4 platform. For example, the total memory available is $112 + 16$ KB (SRAM1 and SRAM2). Moreover, it should be noted that these implementations ensure a runtime independent of any secret key material but are *not* protected against active [3,2] (faults) or passive [22] (side-channel) attacks. For critical applications, such as secure boot on vehicle network processors, protection against these advanced attacks is often a minimal requirement.

We implemented the Dilithium algorithms for all parameter sets from scratch and ensured they comply with the proposed specification and pass the Known-Answer-Tests provided in [26]. Our main focus is on the signature *verification* since this is the only functionality required in the secure boot flow. For verification the protection against passive attacks is not relevant; a side-channel attack

tries to deduce information about the bits of the secret key material used during execution based on, for instance, the observed power consumption of the device. However, no secret key material is used during signature verification. Protection against fault attacks is required since it would be trivial to force acceptance of a wrong signature by introducing a well-targeted fault in the implementation. Our implementation includes countermeasures against single-targeted fault attacks: this is achieved by both adding countermeasures protecting the control flow as well as algorithmic checks to ensure no steps are skipped or memory regions have been altered.

As can be observed from the pqm4 benchmarking framework, the stack consumption of Dilithium is significantly larger compared to the classic public-key counterparts (such as RSA and ECC). The stack requirement for signature verification for Dilithium 3 reported by pqm4 is around 58 KB. Recent work [11] has shown how to reduce this stack to around 10 KB. Our fault attack resistant Dilithium verification code requires less than 3 KB of additional stack for all parameter sets. This is a huge improvement over previous works: still, it is an order of magnitude larger compared to signature verification based on elliptic curves.

There are two variants of Dilithium specified in the supporting documentation: the main version where symmetric primitives for matrix expansion are instantiated with SHAKE, and a second version where AES is used. The latter was included mostly to demonstrate the efficiency of Dilithium on platforms which do not have support for SHAKE yet or have dedicated hardware support for AES. In this work we only focus on the recommended variant using SHAKE. For the SHAKE implementation we use (a slightly modified version of) the assembly code published in the eXtended Keccak Code Package[1] (XKCP).

4.2 Firmware Integration

Given a functional Dilithium implementation, the next step is to update the HSE-FW to support its use. This is made easy by the fact that the Dilithium signature verification API (as mandated by NIST) is virtually identical to that of RSA and elliptic-curve-based signature schemes. The main complications arise from the fact that the memory use of Dilithium is higher, both in terms of key and signature size as well as stack. However, keys still easily fit into the key catalog, while 3 KB stack can be handled by the HSE. Hence we observed no significant obstacles in adding Dilithium support to the boot flow.

In order to evaluate and benchmark the integration, we created a simple demo application. For this purpose we require the compiled application images to be accompanied by a Dilithium signature, for which we wrote a stand-alone command-line tool. This tool was written in C, and was built around the avx2 implementation of Dilithium from the CRYSTALS team.[2] Using our signing

[1] `https://github.com/XKCP/XKCP/blob/master/lib/low/KeccakP-1600/ARM/KeccakP-1600-inplace-32bi-armv7m-le-armcc.s`

[2] `https://github.com/pq-crystals/dilithium`

tool, we pack the compiled application code into a flash image, together with a Dilithium signature and the public key under which the code was signed, and load it together with our demo application image into flash. We also use the demo application as boot loader. On first boot, when secure boot is still disabled, the demo application loads the Dilithium key and signature into the HSE. Following the description in Section 3.2, it installs our application code in a Secure Memory Region using the attached digital signature, and enables secure boot on the device. To verify whether the secure boot configuration was effective, we can query the HSE Core Boot status, which contains the info on which SMRs were correctly verified during boot. In our development setup we do not advance the lifecycle of the device, as that would brick our development setup.

4.3 Performance Results

Beyond validating that a functional Dilithium-based secure boot setup is feasible, it is of course interesting to compare its performance to the status quo. When secure boot is enabled, the boot latency is dominated by signature verification. Therefore, it is sufficient to measure Dilithium verification latency, and compare it to the verification latencies of a selection of other signature schemes.

The latency is not only determined by the choice of signature scheme, but also by the length of the application image. All relevant schemes sign and verify in essentially two steps. First, the variable-length message is *pre-hashed* down to a fixed-size digest, possibly including padding, a public key, a commitment, etc. Afterwards the digest is processed to create the final digital signature. Unfortunately, although this step is essentially independent of the signature scheme, the choice of hash function does slightly differ. For example, for ECDSA [41] a hash function specified in FIPS 180 [39] should be used (e.g., SHA-256) that is applied only on the message itself, while in EdDSA [18] pre-hashing is optional. The Ed25519 instantiation does not pre-hash, reducing the message size implicitly together with a prefix in an application of SHA-512, while Ed25519ph first explicitly reduces the input message using SHA-512. On the other hand, the Dilithium signature scheme signs arbitrary-length messages by hashing them together with the public key, using SHAKE-256. Although the choice of hash function (assuming appropriate length is chosen) is independent of the security of the public-key signature scheme, it can have significant impact on the performance. More concretely, by offering hardware support for SHA-2 and not for (variants of) SHA-3, the S32G274A offers a clear performance benefit for SHA-2. Therefore we investigate two variants: *DilithiumX* for $\mathrm{X} \in \{2, 3, 5\}$ where application images are signed directly with Dilithium, and *DilithiumX-ph* where a SHA-256 hash over the image is signed instead. To investigate the impact of hashing, we measure the verification of a signature on both a small image (1 KiB) and a larger image (128 KiB).

We distinguish between the *installation* time of an application image where the digital signature of Dilithium is verified (the original proof of authenticity) and *boot* time where only the reference proof is verified. Of course, a user can opt

to also verify the proof authenticity on each boot, but the performance impact is large (even in a classical setting), while there are no (significant) security benefits. Because verification of the reference proof does not depend on the choice of digital signature scheme, the boot time is actually not affected by switching to a post-quantum variant. Although the installation for Dilithium is slower than for RSA and elliptic-curve based variants, it is only performed once (or a few times) and its runtime is not as critical. We summarize all of our measurements in Table 1.

Table 1. Latencies of installation (inst.) and boot in milliseconds for supported algorithms on the S32G274A. Key sizes are reported in bytes. The pre-hash (ph) variants of Dilithium first hash the image using SHA-256, and verify the Dilithium signature over the hash.

Alg.	**Size**		**1KiB**		**128KiB**	
	PK	**Sig.**	**Inst.**	**Boot**	**Inst.**	**Boot**
RSA 4K	512	512	2.6	0.0	2.7	0.2
ECDSA-p256	64	64	6.2	0.0	6.4	0.2
Dilithium2	1312	2420	12.1	0.0	158.9	0.2
Dilithium3	1952	3293	17.8	0.0	164.4	0.2
Dilithium5	2592	4595	26.6	0.0	173.3	0.2
Dilithium2-ph	1312	2420	11.1	0.0	11.3	0.2
Dilithium3-ph	1952	3293	16.7	0.0	16.9	0.2
Dilithium5-ph	2592	4595	25.5	0.0	25.7	0.2

From our benchmarks, we see that Dilithium verification of small images is 5–10 times slower than RSA 4K and 2–5 times slower than ECDSA-p256, depending on the chosen post-quantum security level. The security level for Dilithium2, Dilithium3 and Dilithium5 is as at least as high as AES-128, AES-192 and AES-256 respectively, for both classical as well as quantum adversaries, which can help guide in choosing the appropriate security level for a use case.

When verifying small images, the Dilithium signature verification completes in less than 30 ms for all variants. Looking at the results for the verification of larger images without pre-hashing, we see latencies up in the hundreds of milliseconds. As mentioned, this is almost completely attributed to the SHAKE-256 hash has that is applied to the image. With additional SHA-256 pre-hashing, the large-image verification latencies are almost equal to the latencies we measure for small images (actually even lower). It is clear that without hardware support for SHAKE-256, the image verification is dominated by the hashing of the image. In fact, even *with* pre-hashing the dominating cost in Dilithium is the pseudo-random matrix generation using the SHAKE-128 eXtendable Output Function (XOF). Hence, improved latencies for SHAKE variants would significantly help for low-latency signature verification using Dilithium. We do not observe this in

the case of RSA 4K and ECDSA-p256, since they hash the image using SHA-256 (for which hardware acceleration is present). However, we re-iterate that the signature verification only impacts *installation* time of the SMR and is irrelevant for the boot time, for which low latency is much more crucial.

5 Conclusions

The main challenges that can be expected when migrating from classical signature verification schemes such as RSA or ECC to post-quantum variants such as Dilithium, are an increase in memory (keys, signatures as well as stack) and runtime. The significantly larger public keys and signatures do not cause any real practical problems on our target platform in the setting of vehicle network processors. Moreover, we showed that in this setting of signature verification the amount of stack space required for cryptographic operations needs to be increased only marginally. The performance of Dilithium signature verification is indeed worse than that of ECC/RSA verification. However, as this is only performed during installation time, there is no impact on the boot time itself. We believe a transition to post-quantum secure boot can be considered practical for this application.

References

1. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 124–142. Springer, Heidelberg (Sep / Oct 2011). https://doi.org/10.1007/978-3-642-23951-9_9
2. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (Aug 1997). https://doi.org/10.1007/BFb0052259
3. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults (extended abstract). In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (May 1997). https://doi.org/10.1007/3-540-69053-0_4
4. Challener, D., Goldman, K.: Trusted Platform Module Library Specification, Family "2.0", Level 00, Revision 01.59 (2019), https://trustedcomputinggroup.org/work-groups/trusted-platform-module/
5. Cooper, D., Apon, D., Dang, Q., Davidson, M., Dworkin, M., Miller, C.: Recommendation for stateful hash-based signature schemes. SP 800-208, National Institute of Standards and Technology (2020)
6. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (1976)
7. Ding, J., Chen, M.S., Petzoldt, A., Schmidt, D., Yang, B.Y., Kannwischer, M., Patarin, J.: Rainbow. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions

8. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: A lattice-based digital signature scheme. IACR TCHES **2018**(1), 238–268 (2018). https://doi.org/10.13154/tches.v2018.i1.238-268, https://tches.iacr.org/index.php/TCHES/article/view/839
9. Fritzmann, T., Vith, J., Flórez, D., Sepúlveda, J.: Post-quantum cryptography for automotive systems. Microprocessors and Microsystems **87**, 104379 (2021). https://doi.org/https://doi.org/10.1016/j.micpro.2021.104379, https://www.sciencedirect.com/science/article/pii/S0141933121005299
10. GlobalPlatform Technology: Root of Trust Definitions and Requirements Version 1.1 (GP_REQ_025) (2018), https://globalplatform.org/specs-library/globalplatform-root-of-trust-definitions-and-requirements/
11. Greconici, D.O.C., Kannwischer, M.J., Sprenkels, D.: Compact dilithium implementations on cortex-M3 and cortex-M4. IACR TCHES **2021**(1), 1–24 (2021). https://doi.org/10.46586/tches.v2021.i1.1-24, https://tches.iacr.org/index.php/TCHES/article/view/8725
12. Güneysu, T., Krausz, M., Oder, T., Speith, J.: Evaluation of lattice-based signature schemes in embedded systems. In: International Conference on Electronics, Circuits and Systems (ICECS). pp. 385–388. IEEE (2018)
13. Hermelink, J., Pöppelmann, T., Stöttinger, M., Wang, Y., Wan, Y.: Quantum safe authenticated key exchange protocol for automotive application. In: 18^{th} escar Europe : The World's Leading Automotive Cyber Security Conference (Konferenzveröffentlichung) (2020). https://doi.org/10.13154/294-7549
14. Hulsing, A., Bernstein, D.J., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Kampanakis, P., Kolbl, S., Lange, T., Lauridsen, M.M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P., Aumasson, J.P., Westerbaan, B., Beullens, W.: SPHINCS+. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
15. Hülsing, A., Butin, D., Gazdag, S.L., Rijneveld, J., Mohaisen, A.: XMSS: Extended Hash-Based Signatures. RFC 8391 (2018)
16. International Electrotechnical Commission (IEC): Functional safety of electrical/electronic/programmable electronic safety-related systems. IEC 61508 (2010)
17. International Organization for Standardization (ISO): Road vehicles - functional safety. ISO 26262 (2018)
18. Josefsson, S., Luisvaara, I.: RFC 8032: Edwards-Curve Digital Signature Algorithm (EdDSA). Internet Research Task Force (IRTF) (Jan 2017)
19. Kampanakis, P., Panburana, P., Curcio, M., Shroff, C.: Post-quantum hash-based signatures for secure boot. In: Silicon Valley Cybersecurity Conference. Springer (2020). https://doi.org/10.1007/978-3-030-72725-3
20. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. Workshop Record of the Second PQC Standardization Conference (2019)
21. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation **48**, 203–209 (1987)
22. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_25
23. Kumar, V.B., Gupta, N., Chattopadhyay, A., Kasper, M., Krauß, C., Niederhagen, R.: Post-quantum secure boot. In: Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1582–1585. IEEE (2020)

24. Lyubashevsky, V.: Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 598–616. Springer, Heidelberg (Dec 2009). https://doi.org/10.1007/978-3-642-10366-7_35
25. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_43
26. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehlé, D., Bai, S.: CRYSTALS-DILITHIUM. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
27. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_1
28. McGrew, D.A., Curcio, M., Fluhrer, S.R.: Hash-Based Signatures. RFC 8554, RFC Editor (04 2019), https://www.rfc-editor.org/rfc/rfc8554.txt
29. Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO'85. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (Aug 1986). https://doi.org/10.1007/3-540-39799-X_31
30. National Institute of Standards and Technology: Post-quantum cryptography standardization. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization
31. National Institute of Standards and Technology: Digital signature standard (dss) (draft) (October 2019), fIPS PUB 186-5 Federal Information Processing Standards Publication
32. Peikert, C., Regev, O., Stephens-Davidowitz, N.: Pseudorandomness of ring-LWE for any ring and modulus. In: Hatami, H., McKenzie, P., King, V. (eds.) 49th ACM STOC. pp. 461–473. ACM Press (Jun 2017). https://doi.org/10.1145/3055399.3055489
33. PKCS #1: RSA cryptography standard. RSA Data Security, Inc. (Sep 1998), version 2.0
34. Ravi, P., Gupta, S.S., Chattopadhyay, A., Bhasin, S.: Improving speed of Dilithium's signing procedure. In: Belaïd, S., Güneysu, T. (eds.) Smart Card Research and Advanced Applications – CARDIS. LNCS, vol. 11833, pp. 57–73. Springer (2019). https://doi.org/10.1007/978-3-030-42068-0_4
35. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005). https://doi.org/10.1145/1060590.1060603
36. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the Association for Computing Machinery **21**(2), 120–126 (1978)
37. Sanwald, S., Kaneti, L., Stöttinger, M., Böhner, M.: Secure boot revisited: challenges for secure implementations in the automotive domain. SAE International Journal of Transportation Cybersecurity and Privacy **2**(11-02-02-0008), 69–81 (2020)
38. Certicom research, standards for efficient cryptography group (SECG) — sec 1: Elliptic curve cryptography. http://www.secg.org/secg_docs.htm (Sep 20, 2000), version 1.0
39. Secure hash standard. National Institute of Standards and Technology, NIST FIPS PUB 180-4, U.S. Department of Commerce (Aug 2015)

40. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th FOCS. pp. 124–134. IEEE Computer Society Press (Nov 1994). https://doi.org/10.1109/SFCS.1994.365700
41. Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). American National Standards Institute (ANSI), X9.62-1998 (Nov 2015)