IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 2, pp. 714–734. DOI:10.46586/tches.v2024.i2.714-734

# TPMScan: A wide-scale study of security-relevant properties of TPM 2.0 chips

Petr Svenda[1], Antonin Dufka[1], Milan Broz[1], Roman Lacko[1], Tomas Jaros[1], Daniel Zatovic[2] and Josef Pospisil[3]

[1] Masaryk University, Brno, Czech Republic, <svenda@fi.muni.cz>
[2] Red Hat, Brno-Medlánky, Czech Republic
[3] National Cyber and Information Security Agency, Brno, Czech Republic

**Abstract.** The Trusted Platform Module (TPM) is a widely deployed computer component that provides increased protection of key material during cryptographic operations, secure storage, and support for a secure boot with a remotely attestable state of the target machine. A systematic study of the TPM ecosystem, its cryptographic properties, and the orderliness of vulnerability mitigation is missing despite its pervasive deployment – likely due to the black-box nature of the implementations. We collected metadata, RSA and ECC cryptographic keys, and performance characteristics from 78 different TPM versions manufactured by 6 vendors, including recent Pluton-based iTPMs, to systematically analyze TPM implementations. Surprisingly, a high rate of changes with a detectable impact on generated secrets, the timing of cryptographic operations, and frequent off-chip generation of Endorsement Keys were observed. Our analysis of public artifacts for TPM-related products certified under Common Criteria (CC) and FIPS 140 showed relatively high popularity of TPMs but without explanation for these changes in cryptographic implementations.

Despite TPMs being commonly certified to CC EAL4+, serious vulnerabilities like ROCA or TPM-Fail were discovered in the past. We found a range of additional unreported nonce leakages in ECDSA, ECSCHNORR, and ECDAA algorithms in dTPMs and fTPMs of three vendors. The most serious discovered leakage allows extraction of the private key of certain Intel's fTPM versions using only nine signatures with no need for any side-channel information, making the vulnerability retrospectively exploitable despite a subsequent firmware update. Unreported timing leakages were discovered in the implementations of ECC algorithms on multiple Nuvoton TPMs, and other previously reported leakages were confirmed. The analysis also unveiled incompleteness of vulnerability reporting and subsequent mitigation with missing clear information about the affected versions and inconsistent fixes.

**Keywords:** TPM · RSA · ECDSA · ECSCHNORR · ECDAA · ECC key recovery

## 1 Introduction

Five different types of TPMs are widely deployed today [KK19], all implementing parts of the same TPM 2.0 specification but designed for different applications and platforms. Discrete TPMs (dTPMs) are implemented as standalone hardware modules, placed in a socket, with some level of anti-tampering measures. Firmware TPMs (fTPMs) are implemented in the CPU's firmware, usually running on a separate physical core from the CPU cores. Integrated TPMs (iTPMs) are implemented as dedicated hardware integrated into the CPU package – they are the middle ground between dTPMs and fTPMs. The remaining two types, TPM simulators and virtualized TPMs, are not studied in this work.

Despite the wide prevalence of TPMs in current machines and their frequent evaluation by thorough (and expensive) security certification under the Common Criteria scheme

Received: 2023-10-15 Accepted: 2023-12-15 Published: 2024-03-12

aimed typically at the EAL4+ level, serious security vulnerabilities like ROCA [NSS+17] (practical factorization of RSA keys due to a specific format of generated primes) or TPM-Fail [MSEH20] (private ECC key extraction via timing side-channel leak present in signature nonces) were discovered and reported. While vendors reacted by issuing security advisories and patching the discovered vulnerabilities, the extent of systematic mitigation is unclear due to the unavailability of manufacturers' TPM firmware source code. We aim to fill this gap by analyzing a wide range of TPM firmware versions in real TPMs for known and new vulnerabilities in RSA and ECC-based algorithms implementations. As the manufacturer TPM source code is not publicly available, we study information extracted from security certificates, CVE records, security advisories and bulletins and, most importantly, directly from the externally observable changes and cryptographic properties of a large dataset of TPMs as a proxy for security-relevant code changes. Such analysis provides more insight into the otherwise proprietary ecosystem, helping to mitigate existing overlooked vulnerabilities and incomplete fixes, and establish a basis for further security studies.

**Paper contribution:**

- An extensive open dataset of properties and cryptographic metadata collected from 78 different dTPMs, fTPMs, and iTPMs spanning more than ten years.
- Discovery of several new leakages of nonce bit(s) in ECC algorithms, including a range of older Intel fTPMs retrospectively exploitable with only nine signatures and no side-channel information required.
- Analysis of black-box RSA and ECC cryptographic implementations, documenting the surprisingly high rate of changes, Endorsement Key injection, and inconsistent vulnerability handling for some vendors.
- Mapping of support of standard cryptographic operations and ones required for more advanced cryptographic protocols.

The data collection and processing tooling with a database of all collected results are available under a permissive license at https://crocs.fi.muni.cz/papers/tpm_ches2024. We sincerely thank all the people who contributed to the data collection. The new vulnerabilities were responsibly disclosed to the corresponding vendors.

The paper is organized as follows: The rest of Section 1 discusses related work. Insight into CC/FIPS140 certification artifacts of TPM products is provided in Section 2. Section 3 describes the TPM data collection approach and the resulting dataset. Sections 4 and 5 analyze ECC and RSA implementations, respectively. Algorithm support and performance are surveyed in Section 6, including utilization for black box analysis of Microsoft Pluton-based iTPMs. The conclusions follow in Section 7.

## 1.1 Related work

TPMs are present on the majority of modern computers and mobile devices and are starting to be required by the latest versions of operating systems, where they serve as roots of trust for trusted computing applications. The first specification describing Trusted Platform Modules was published in 2001 by the Trusted Computing Platform Alliance (TCPA) [TCP01]. The Trusted Computing Group (TCG), formed from the TCPA in 2003, standardized the successor of this specification in 2009 as the Trusted Platform Module Specification 1.2 [TCG09]. TPM 1.2 specification offered a one-size-fits-all solution with a single hierarchy, single root key, a limited set of cryptographic algorithms (only SHA1, RSA, HMAC, and MGF1 being mandatory), and very limited options for advanced applications. This version targeted computers and servers and was unsuitable for more constrained environments like mobile computing, IoT, or industrial applications. For this reason, TCG released TPM 2.0 [TCG19b], with a platform-specific specification setting different requirements for different use cases, like desktop PC or mobile.

With the TPM 2.0 specification, TPMs started being usable also for a wider range of applications. Chen and Urian [CU16] have summarized the interface provided by the TPM 2.0 specification that can be used for implementing advanced protocols with keys stored securely inside TPMs. These building blocks can be used to construct protocols that perform operations with the secret key securely within the TPM but rely on the host for privacy and other protocol properties. A number of remote attestation protocols with better privacy properties have been proposed [CL13, PZ11, CU15]. Chen and Urian [CU16] have also shown how four ECC-based asymmetric decryption algorithms can be implemented with TPMs: ECIES, PSEC, ACE, and FACE. Multi-party signature protocols have been implemented on TPMs for BLS signatures [HA20] and even for a restricted variant of Schnorr signatures [Jan23].

The TPM's wide adoption and critical use case as a root of trust created a need for formal certification. The TCG maintains a certification program for TPMs based on Common Criteria [TCG17, TCG23a]. Still, many devices from the certified list [TCG23b], including CC EAL4+ certified ones, were found vulnerable to various exploits, including flaws in the construction of primes for RSA keys (the ROCA attack), leading to a significant loss of entropy [NSS+17] or allowing for qualified guessing of their origin [SNS+16]. Other works [JSSS20, SETA21] exploited noisy leakage of bit-length of ECDSA nonces to extract the ECC private key. The challenges that the CC certification faced were expressed already around 2004 by Hearn [Hea04] and eight years later by Murdoch et al. [MBA12], who discuss why both CC and FIPS 140 fell short of their promise, pointing to the lack of transparency. Beckert et al. [BGG10] highlight that formal verification of only the specification, but not the implementation, is required to achieve higher EALs. Many of the schemes' problems are also mentioned in a user study [HTAP18] of 29 users trained in the NIST cryptographic validation program. A study by Kaluvuri et al. [KBR14] performs a data-driven analysis of EAL4+ ICs and smartcard CC certificates. At the time of the [KBR14] study, no TPM 2.0 device was certified.

Many other attacks on TPMs exist, most commonly targeted at dTPMs and fTPMs. The dTPMs are commonly vulnerable to physical attacks on their exposed data bus, such as sniffing (e.g., sniffing of Bitlocker VMK on the SPI[1], LPC[2], or I2C[3]) or MITM attacks. Inserting a TPM Genie-type device between the socket and dTPM allows an attacker to hijack communication packets, forge the TPM's RNG output, and attack PCR extensions used in attestation, secure boot, and sealing [Jer18]. The fTPMs and iTPMs are implemented either in the CPU's firmware (AMD and Intel) or as dedicated hardware integrated into the CPU package (e.g., Pluton-enabled CPU[4] starting from Ryzen 6000 and Qualcomm Snapdragon 8cx Gen 3 CPUs). As they do not have any physically exposed pins or communication interfaces, attacks on them focus mostly on side channels, fault injection, and firmware exploits. FaulTPM [JWBS23] is a voltage-glitching fault injection attack compromising the entire internal state of AMD Zen 2/3 fTPMs. TPM-Fail [MSEH20] is a private key recovery attack using leakage of secret-dependent execution times during the generation of signatures based on elliptic curves, affecting certain Intel fTPMs and STM dTPMs certified up to the CC EAL4+ security level.

This work focuses primarily on the security analysis of RSA and ECC implementations.

[1] https://github.com/denandz/lpc_sniffer_tpm/.

[2] https://github.com/nccgroup/TPMGenie.

[3] https://astralvx.com/stealing-the-bitlocker-key-from-a-tpm/.

[4] https://learn.microsoft.com/windows/security/hardware-security/pluton/pluton-as-tpm.

## 2 TPMs in CC/FIPS certification

As TPMs are often used in environments imposing strict security requirements, they may need to undergo a security certification process to comply with existing regulations. To shed more light on the current state of the TPM ecosystem from the perspective of security certifications, we analyzed public certification documents released under Common Criteria and FIPS 140 certification schemes with the help of the *sec-certs* project [JJS+23].

To perform the analysis, we first downloaded all public CC and FIPS certification documents. As certification documents are not designed for automated processing, the source PDFs were first converted to text files. Subsequently, we filtered documents that mentioned TPM in its title or multiple times in the document's body to avoid false positives[5]. Furthermore, we had to manually filter out some documents that were coincidentally using the TPM abbreviation for a different technology or only included TPM as a component.

For each matched TPM certificate, we identified the TPM specification version (1.2 or 2.0) and the revision it complies with. We did this by matching regular expressions of version numbers against the text content of the documents, and in case we did not find enough clear matches in some certificates, we manually annotated them. For the rest of the analysis, we used only TPM 2.0 certificates. We identified 85 such certification documents (58 Common Criteria and 27 NIST FIPS 140-2/3) by four vendors. All CC certificates were issued at EAL4+; FIPS certificates mostly targeted security level 2.

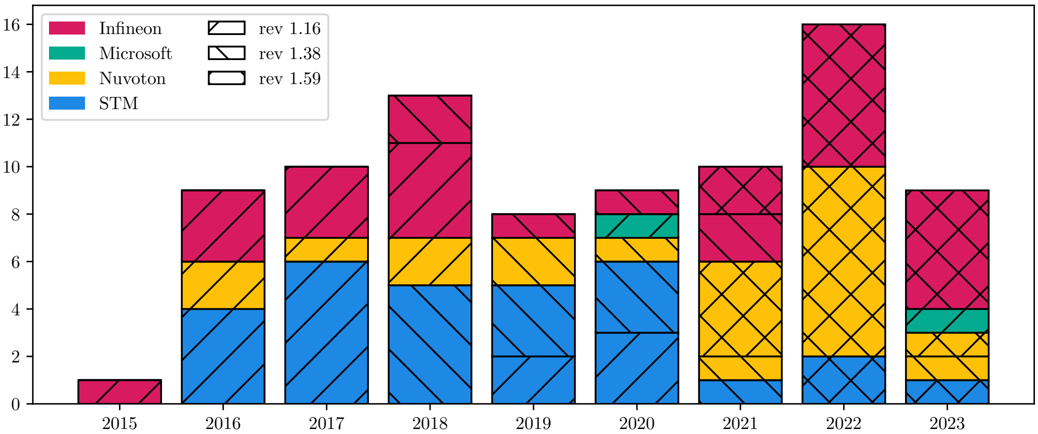


**Figure 1:** Number of TPM 2.0 certificates issued to vendors by year. The specification revision the certified TPM complies with is shown with a bar pattern.

Figure 1 shows that TPM 2.0 devices started being certified in 2015, with the first certified TPM by Infineon, who, to this day, published 30 TPM 2.0 certificates. STM had its first TPM 2.0 chip certified a year later, but since then, they have also obtained 30 TPM 2.0 certificates. Nuvoton has 23 certified TPM 2.0 chips. Lastly, Microsoft has two virtual TPMs certified under the FIPS scheme. Neither Intel and AMD fTPMs nor Pluton-based iTPMs are currently certified under any of the schemes.

The first TPMs certified with revision 1.38 started appearing in 2018, two years after the revision had been published. Revision 1.59 started appearing in 2021, again, two years after its publication. We did not see any certificate document mentioning versions of the TPM 2.0 specification lower than 1.16.

The documents of selected TPMs were inspected for information on changes in cryptographic libraries with respect to discovered vulnerabilities in earlier versions. However, we did not find this information present in any of the public documents.

[5] Some documents only mention TPM technology as a side note or in references.

# 3 TPM data collection and processing

We developed two TPM analysis tools for Linux (`tpm2-algtest`) and Windows (`tpm_pcr`) OSes and collected TPM data from three principal sources: university-managed machines, a software-compatibility testing cluster, and end-user machines of volunteers. While we also collected some results for legacy TPM 1.2 chips, we focused primarily on version 2.0.

The simpler `tpm_pcr` tool collects only PCR register values[6], system information, and Endorsement and Storage Root Keys on Windows OS, with its implementation being based on Microsoft's PCPTool[7]. The data collection typically takes around 10 seconds and can be scheduled to run periodically (e.g., once per day). While allowing for easy and quick data gathering by volunteers, the collected properties are limited.

The `tpm2-algtest` tool collects all the persistent and temporal characteristics of a TPM on Linux OS. The implementation relies on the `tpm2-tools` library[8] to communicate with the TPM via Linux kernel drivers. The tool is also available in the form of a Fedora-based bootable image with a custom wizard interface for controlled data collection on any computer, independently of the installed OS. The tool's runtime depends heavily on the number of measured operations and the speed of the target TPM chip, taking between tens of minutes (typically fTPMs) up to ten hours (dTPMs) in the default configuration.

The tool collects public keys of freshly generated RSA and ECC key pairs for all relevant algorithms and key lengths, and the corresponding private keys. The key pairs are created using the `TPM2_Create` command, which outputs the public key in plain and the private key encrypted. To extract the private key, we generated all key pairs with a policy allowing for duplication that enables the key to be used with the `TPM2_Duplicate` command. When this command is called on the generated key with *newParentHandle* and *symmetricAlg* parameters set to `TPM2_ALG_NULL`, the encryption is disabled, and the private key is also output in plain. A summary of collected properties is listed in Table 1.

**Table 1:** TPM properties collected by `tpm2-algtest`. The number of samples listed is per single collection run and can be increased by configuration option or by repeated runs.

| **Persistent properties** | # | **Temporal properties** | # |
|---|---|---|---|
| System info | — | $PCR_0$–$PCR_{23}$ values | — |
| TPM capabilities | — | RSA & ECC on-chip gen. keys | 1000x |
| Algorithms performance | 1000x | RSA & ECC signatures | 1000x |
| Endorsement Key / SRK | 2B+2B | RNG output | 512kB |

As `tpm2-algtest` has been developed in parallel with the collection of data samples, some collected data may be missing for some TPM firmware. In particular, ECC keys and operation performance are collected for 45 out of 61 firmware versions since the functionality was included only in 2022, and we could not re-run the data collection for a small number of older, community-provided results.

We focus on the analysis of the following TPM-related properties:

(i) persistent properties, like algorithm support and operations performance and data timing dependency of different TPM chips (different vendor and/or firmware version),

(ii) variations between physical chips of the same type (same vendor and firmware version), caused, e.g., by infrequent malfunctioning or manufacturing variability,

(iii) observed changes caused by a different firmware version (same vendor but different firmware versions, possibly also different underlying TPM hardware),

[6] Memory registers with a hash chain of configuration data and binaries executed.
[7] https://github.com/microsoft/TSS.MSR/tree/master/PCPTool.v11.
[8] https://github.com/tpm2-software/tpm2-tools.

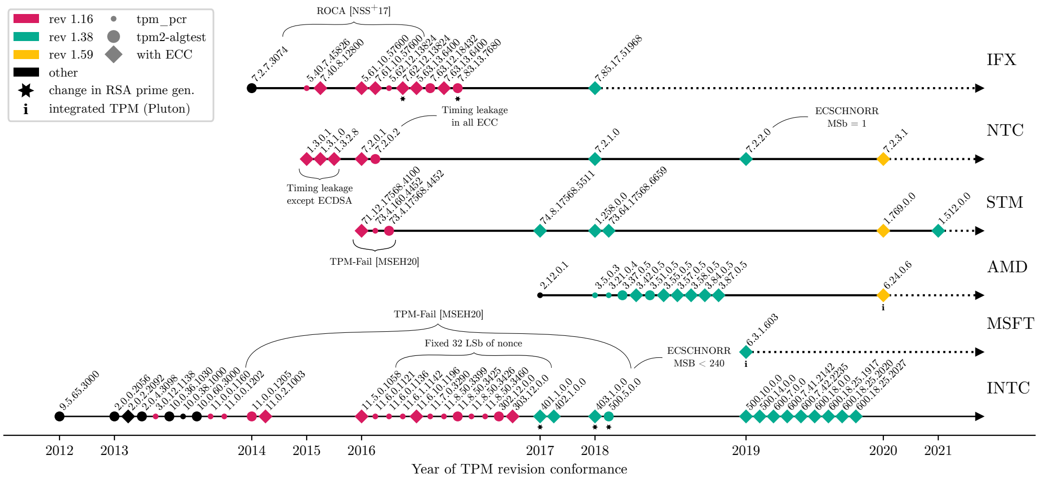


**Figure 2:** The collected dataset annotated with changes observed in cryptographic implementations, specification revisions, and with detected vulnerabilities. The listed year is of the errata the firmware conforms to, not necessarily the year of the firmware release, and is typically 1-2 years behind a release of the actual platform with such a TPM chip. The year value for measurements from `tpm_pcr` was interpolated from related firmware versions. Vulnerabilities in ECC implementations were verified only on samples with ECC data collected. Vulnerabilities without a citation are newly found ones.

(iv) properties of TPM-unique Endorsement Keys and Storage Root Keys,

(v) properties of cryptographic material produced by target TPM like generated keys, signatures, and random data.

All the operations executed and measured during data collection are within the temporary, transient key hierarchy [TCG19a], which is erased during any reboot by the TPM itself. We do not perform any firmware updates. We do not collect any personally identifiable data, no attestation keys, or user-specific content of the non-volatile TPM memory (NVRAM) and only the first and the last two bytes of an Endorsement Key. All tools are available under a permissible open-source license.

## 3.1 Dataset from 78 unique TPM firmware versions

The dataset was collected over the period of seven years and contains 78 unique firmware produced by 6 different vendors: Infineon (IFX), Nuvoton (NTC), STMicroelectronics (STM), Advanced Micro Devices (AMD), Microsoft (MSFT), and Intel (INTC). A summary of collected firmware versions is shown in Figure 2, with the most recent versions released till the middle of 2023, including Pluton-based iTPMs. Only a few versions certified in 2023 (STM, IFX) are missing, as they are not yet available on end-user platforms. We executed two primary data collection scenarios:

1. **Many computers with different TPM chips** (different vendor and/or version), collected with `tpm2-algtest` tool on 200+ computers, mostly by volunteers and on the testing cluster. This case mainly examines properties (i), (iii), (iv), and (v).
2. **Many computers with the same TPM chip** (same vendor and version), preferably also with the same OS environment. This scenario primarily examines property (ii). Properties (i) and (v) are also addressed with orders of magnitude more data, but only for two specific TPM chips (IFX 5.63.13.6400 and NTC 7.2.2.0). Property (iv)

was also analyzed as different TPM chips from the same vendor and firmware version should still have unique EKs. Data was collected with `tpm2-algtest` and `tpm_pcr` tools on 248 managed university computers and 100+ computers from volunteers.

The majority of TPM firmware in our dataset claims compliance with revision 1.16 (35x), followed by 31 samples of the firmware of revision 1.38, and lastly, three firmware samples of the latest revision 1.59, originally published in 2019. The TCG website provides a specification of five TPM 2.0 revisions: 0.96, 0.99, 1.16, 1.38, and 1.59, but we have also encountered some TPMs declaring compliance with revisions that we could not find in public sources: 0.93 (1x), 1.03 (6x), 1.15 (1x), and 1.21 (1x).

Note that as volunteers contributed some of the measurements, we do not always have full control over the measurement platform, which may exhibit higher variability of measurements due to other tasks executed on the platform. We cross-verified results for the same TPM version where multiple measurements were available, and we did not observe any "unexplainable" deviations in the presented results. The sanitized dataset, atop the tooling, is openly released to facilitate reproducibility and further research[9].

# 4 Vulnerabilities in ECC implementations

A detectable bias in ECC private key or signature nonce can decrease its security level and, in some cases, even lead to full private key recovery. Such an attack on TPMs has been previously demonstrated by the authors of TPMFail [MSEH20], which has exploited a timing side-channel present in scalar multiplication in several Intel and STM TPMs.

To identify this class of problems, we collected at least 1000 key pairs and signatures together with the time required to generate them for all supported combinations of elliptic curves and signature algorithms on 45 unique TPM firmware versions. As we collected private keys corresponding to the issued signatures, we were able to extract nonces used by the signing algorithm. All this information was subsequently subjected to randomness analysis to identify static biases, and timing analysis to look for the implementation's non-constant behavior.

The nonce extraction formulas were derived from the signing algorithm used to create the corresponding signature [TCG19b]:

$$\text{ECDSA: } s^{-1}(e+rx) \pmod{n}, \qquad \text{ECSCHNORR: } s-rx \pmod{n}$$
$$\text{SM2: } s+sx+rx \pmod{n}, \qquad \text{ECDAA}^{10}\text{: } s-H(r||e)x \pmod{n}$$

where $(r,s)$ are signature components output by the signing algorithm, $x$ is the private key, $e$ is the signed challenge, $n$ is the order of the used elliptic curve, and $H$ is the used hash function.

To ensure that we used the correct approach, we first verified each signature with the corresponding verification algorithm. As it turned out, not all TPM implementations comply with the specifications they claim, which we had to reflect during nonce extraction. The ECSCHNORR computation has changed[11] in revision 1.33, and the ECDAA computation in revisions 1.35 and 1.36, but some of these changes were already included in TPMs claiming to comply only with revision 1.16.

In particular, INTC 303.12.0.0 implements the new ECSCHNORR and ECDAA algorithms, IFX 7.63.13.6400 and 5.63.16.6400 implement the new ECDAA, and NTC 7.2.0.1

[9] Paper supplementary materials, database of results and open tools are available at `https://crocs.fi.muni.cz/papers/tpm_ches2024`.

[10] This formula was used only for ECDAA from TPMs complying with the specification revision 1.36 or higher; otherwise, the ECSCHNORR approach was used as it corresponds to the original computation.

[11] This change does not affect the nonce extraction of ECSCHNORR signatures, as the difference is only present in the challenge computation, and that value is directly output by the signing algorithm as $r$.

and 7.2.0.2 implement the new ECSCHNORR algorithm. Additionally, we discovered that NTC 7.2.1.0 and 7.2.2.0 do not seem to produce valid ECDAA signatures according to any published revision[12].

## 4.1 ECC secret's randomness

Typical randomness testing tools like NIST STS [BRS+10], Dieharder [BEB18], and TestU01 [LS07] are primarily designed for analyzing large amounts of data, but they do not work well with limited data sources like ours. Therefore, to identify static biases present in the generated keys and nonces, we used the `booltest` tool [SKKS19], which is more suitable for performing statistical tests on a low amount of data. The tool was run on concatenations of all generated keys and nonces[13] in the default configuration, in which we only adjusted the block size to correspond to the bit-length of analyzed keys and nonces.

No statistically significant deviations were found in private keys, which was expected, as the algorithm for ECC key generation is clearly defined in the TPM specification [TCG19b], and any observable deviation would imply diverging from the specified key generation. However, the analysis tool pinpointed biases in nonces generated by a number of Intel firmware and one Nuvoton firmware. Upon further inspection, we discovered the following three new problems:

1. The lowest four bytes of nonces of ECDSA and ECSCHNORR algorithms were set to `0x00000001` (INTC 11.5.0.1058, 11.6.10.1196, and 303.12.0.0).
2. The most significant byte of ECSCHNORR nonce never exceeded 240 (INTC 403.1.0.0).
3. The most significant bit of ECSCHNORR nonce was always set to 1 (NTC 7.2.2.0).

The first problem is the most severe and leads to easily exploitable key extraction vulnerability. It sets 32 least significant bits to a fixed value, which leaks a lot of information about the private key when used in a signature computation. By following approaches used in earlier works [MSEH20, JSSS20], we implemented an attack on this type of information leakage utilizing the LLL algorithm [LLL82] to solve the hidden number problem [BV96]. In our experiments, we were able to reconstruct the private key using *only* 9 signatures with no need for active observation of the signing process. For comparison, the TPM-Fail attack [MSEH20] needs *active* observation of the creation of around 1300 signatures. Coincidentally, this issue is present on TPM firmware versions that also suffered from the TPM-Fail vulnerability [MSEH20] and was fixed after revision 403.1.0.0. Still, this attack can have severe consequences for systems with outdated firmware or systems that created signatures with vulnerable versions in the past, as mere ex-post availability of 9 signatures leads to the reconstruction of the used private key and subsequent signature forgery.

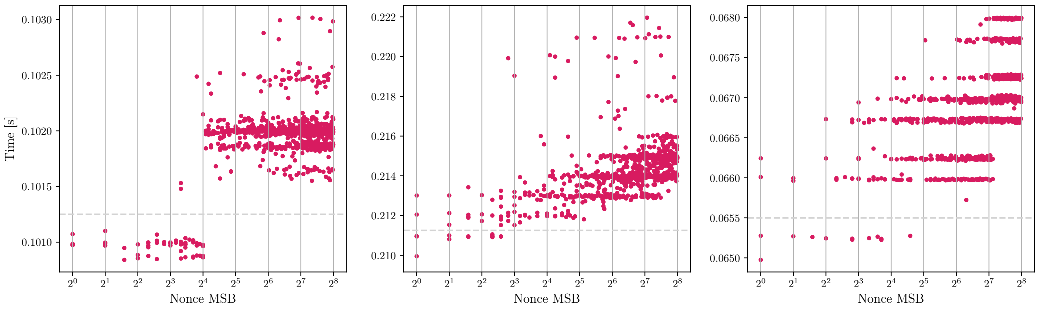
The other two issues leak only one or fewer bits of information, which is too little to practically attack schemes on 256-bit curves with published approaches and a realistically small number of obtained signatures. Still, the presence of such biases signifies a lack of implementation testing and should have been discovered at an earlier stage or at least during certification evaluation (Nuvoton's ANSSI-CC-2020/21 certificate for 7.2.2.0).

## 4.2 ECC operation timing

As the typical timing side-channel attacks [JSSS20, MSEH20] utilize the time dependency of scalar multiplication on the significant bit-length of the used scalar, we plotted the computation time of key generation and signing against the most significant byte of a given private key and nonce, respectively.

[12] It might be compliant with ECDAA of revision 1.35, but we did not find it published.

[13] We always kept keys and nonces from different curves and algorithms separated.



INTC 303.12.0.0 STM 71.12.17568.4100 NTC 1.3.2.8

**Figure 3:** The signing time dependency on the most significant byte of nonce using the ECDAA algorithm on the `BN_P256` curve on TPM firmware by Intel, STM, and Nuvoton. Signing time reveals nonces' topmost bits (e.g., zero bits in samples below the dashed line). This information can be exploited by a LLL-based method to recover the private key.

The plots (Figure 3 shows selected samples) revealed an observable time dependency in implementations of three TPM vendors: Intel, STM, and Nuvoton. For the former two, this was a mere confirmation of the previous work [MSEH20], but for Nuvoton TPMs, this vulnerability has been reported only partially.

The dependency was observable for all key generation and signing configurations for Intel and STM firmware[14], which was expected, as the algorithms are likely to be using the same scalar multiplication implementation. However, it was not the case for Nuvoton firmware 1.3.0.1, 1.3.1.0, and 1.3.2.8, where ECDSA did not exhibit the time dependency, while ECDAA, ECSCHNORR, and key generation algorithms did. Therefore, we presume that Nuvoton's ECDSA implementation was using a different, constant-time implementation of scalar multiplication. These are new findings that have not been reported in previous research or security bulletins.

Furthermore, the Nuvoton firmware 7.2.0.1 had been reported to have an observable timing dependency in ECDSA signing[15], yet we noticed the same dependency also in other algorithms (key generation, ECSCHNORR, and ECDAA). We have observed this only with computations on the `NIST_P384` curve, but it is likely also present on 256-bit curves, only below the measurement resolution we obtained for that particular sample.

# 5 Analysis of RSA implementations

We analyze black-box RSA implementations to find the rate of observable changes in TPM cryptographic libraries and the origin of Endorsement Keys for specific TPM version.

We utilize fingerprinting techniques developed in [SNS+16] and extended in [JNS+20]. Although both [SNS+16, JNS+20] used hundreds of thousands or even millions of keys, 1000 keys are generally sufficient to identify one out of distributions previously detected by [JNS+20]. If multiple measurements for the same TPM chip version are available, we aggregate keys together to produce a less noisy overall analysis.

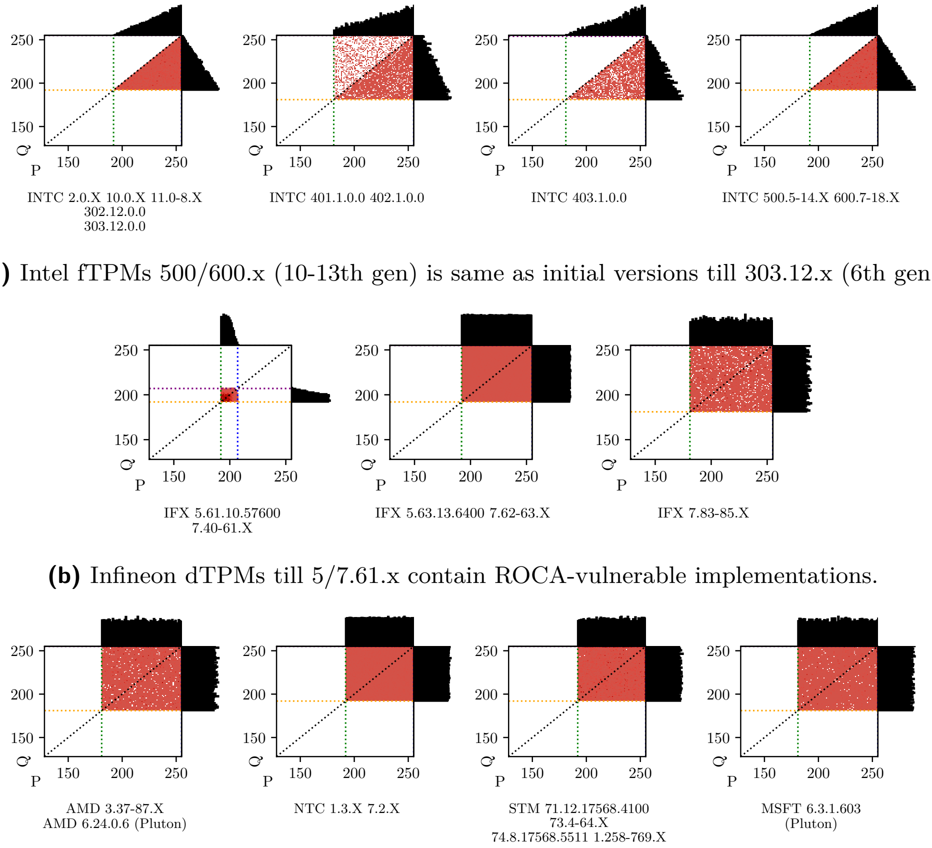
Where private keys were available, [JNS+20] identified four *features* extracted from every private key and used to distinguish between different implementations: (1) the most significant bytes of both $p$ and $q$ (influenced by different strategies to maintain modulus of expected length), (2) existence of small factors between 5 and 17863 in $p-1$ and $q-1$, (3) second least significant bit of both $p-1$ and $q-1$ indicating Blum integers

[14] We detected the leakage in Intel firmware 11.0.2.1003, 11.5.0.1058, 11.6.10.1196, 303.12.0.0, 401.1.0.0, 402.1.0.0, 403.1.0.0, and STM firmware 71.12.17568.4100.

[15] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-25082.

(together with the second feature) believed to be slowing older factorization methods), (4) ROCA-specific fingerprint in public key (Infineon's RSA prime generation algorithm used from year 2004 to 2017 found to be vulnerable to practical factorization [NSS+17]).

For every batch of 1000 keys from a given TPM firmware version (or more if available), we computed the fingerprinting properties and observed if subsequent TPM versions exhibit the same characteristics or if any difference is detected, signaling a change in the underlying RSA key generation implementation.



**(a)** Intel fTPMs 500/600.x (10-13th gen) is same as initial versions till 303.12.x (6th gen).

**(b)** Infineon dTPMs till 5/7.61.x contain ROCA-vulnerable implementations.

**(c)** No changes in AMD fTPMs (including Pluton-based 6.24.0.6 iTPM), Nuvoton dTPMs, and STM dTPMs were observed. Only single version of recent Microsoft's iTPM was available.

**Figure 4:** Observable differences in the RSA prime generation algorithm among firmware versions, implying a change in the underlying implementation. A heatmap (red triangle or square) is computed using the most significant bytes (MSB) of prime $p$ and $q$ from all keypairs for a given version with the shape determined by choice of a prime generation algorithm. The histogram of each prime's MSB is separately visualized on the sides.

## 5.1 Observed changes in RSA prime generation algorithm

We observed significant differences in the most significant bytes of primes $p-1$ and $q-1$ (feature 1) and their evolution in time as visualized in Figure 4. We did not encounter any TPM 2.0 implementation avoiding smaller factors in $p-1$ or $q-1$ (feature 2) or generating Blum integers consistently (feature 3). Only IFX TPMs till versions 5/7.61.10.57600 exhibited the ROCA fingerprint (feature 4).

For Intel fTPMs (Figure 4a), four different ranges are distinguishable, following the evolution of Intel's CPU. The initial implementation, used in 5th gen Braswell, 6th gen Skylake and 7th gen Kaby Lake (2015-2017, 2/3/10/11/302/303.x), was changed twice in the 8th gen Coffee/Whiskey Lake (2017, 401/402.x) and (2018, 403.x), finally ending with observably the same RSA implementations from the 10th gen Comet Lake (2020, 500.x) to the newest 11-13th gens Tiger, Alden and Raptor Lake (2020-2022, 600.x). No version from 9th gen Coffee Lake Refresh was observed, possibly being the same as the 8th gen Coffee Lake (403.x).

For Infineon dTPMs (Figure 4b), three different ranges are distinguishable: versions containing ROCA vulnerability and certified till 2016 (5/7.61.10.57600), then versions till 2018 (7.63.x, likely only with ROCA-related RSA key generation fix[16]), and the most recent range staying the same since 2018 (7.83.x). Infineon additionally certified a range of versions for 1.59 TPM revision during the year 2023, which are so far missing from our analysis due to their unavailability.

No observable changes in RSA keys properties were detected during each vendor's timeline for AMD fTPMs (Ryzen 4000 and 5000 series), AMD iTPM (Ryzen 6000, Pluton-based), Microsoft iTPM (Ryzen 6000, Pluton-based), Nuvoton dTPMs, and STM dTPMs as visualized on Figure 4c. From the recently certified versions, only STM 9.256 from the year 2023 is missing.

Due to a potentially incomplete list of versions available and analyzed, the range boundaries may be imprecise – the interval may extend to other numerically close, uninspected version(s). Less likely, the detected interval may be further split into more sub-intervals if the implementation would change and then change back to the original for the version we did not have in our dataset (e.g., change for only a single version).

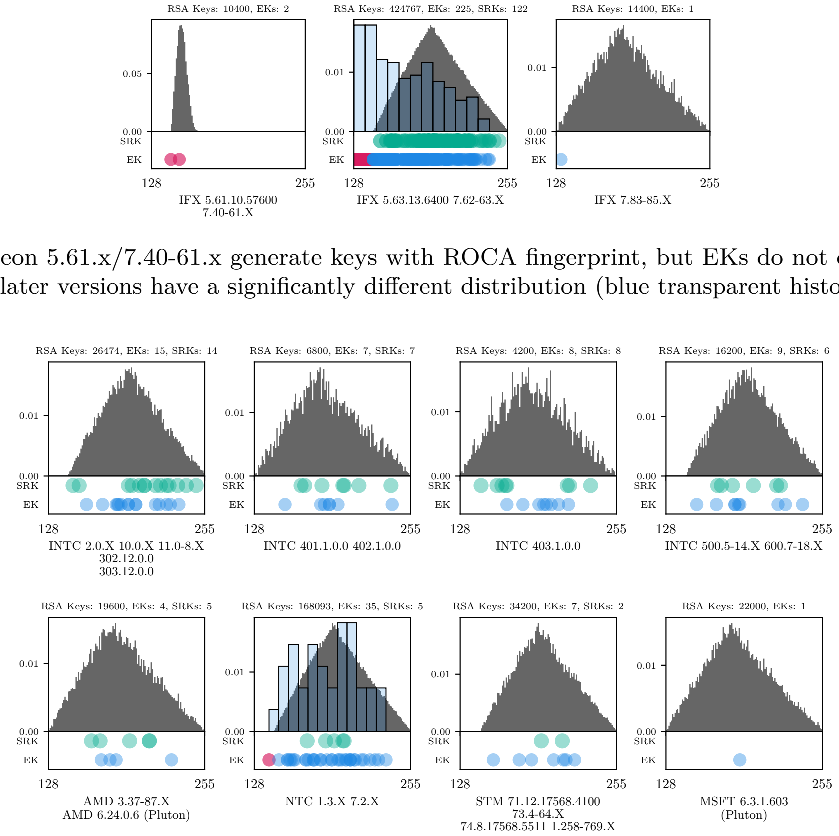
## 5.2 Detection of off-chip Endorsement Key generation

We assess if Endorsement Keys (EKs) were generated directly on a TPM chip (and the corresponding private key never leaves the chip) or first generated outside the TPM (e.g., in a manufacturer's HSM) and only then injected into the TPM during the manufacturing procedure. Initial versions of TPM specification [TCG09] mandated on-chip EK generation, while later revisions allowed for injection from outside environment due to manufacturing process reasons. Nowadays, the EKs are almost always RSA-2048, sometimes complemented with the ECC Endorsement Key. The on-chip RSA private key generation procedure is relatively lengthy and with significant natural variability caused by the prime generation algorithm. While dTPM generates a single key pair in seconds on average, it can take minutes or longer before suitable primes are found. As the dTPMs are produced in batches, the whole batch needs to wait until the last EK is generated before proceeding to the next step, slowing the whole process significantly. This issue was the motivation behind easing the on-chip key generation requirement by TPM specification, but off-chip generation also increases the attack surface for the key compromise.

The RSA key fingerprinting methods described in [SNS⁺16] and [JNS⁺20] provided means for assessing if a given key is (likely) generated by a given cryptographic implementation using an observable difference in the generated key resulting from different implementation choices of the prime generation algorithm. We utilize this approach, but with several limitations when applied to the assessment of EKs.

As only the public part of EK is exportable from TPM, we cannot utilize a more precise fingerprinting method from [JNS⁺20], which requires a private key. Additionally, due to privacy reasons, we do not collect a whole EK in community-provided contributions, as it would serve as a unique identifier of a given machine due to its (typical) immutability. Finally, we typically have only one or very few EKs for every TPM version tested and not

[16] Version 7.62.x was CC certified notably fast in only 6 months after the private vulnerability notification.



**(a)** Infineon 5.61.x/7.40-61.x generate keys with ROCA fingerprint, but EKs do not exhibit it. EKs for later versions have a significantly different distribution (blue transparent histogram).

**(b)** Intel fTPM, AMD fTPM, and STM dTPM show no signs of injected EKs. One injected EK was detected for Nuvoton dTPM.

**Figure 5:** Distribution of the most significant byte of Endorsement Keys (blue circles) and Storage Root Keys (green circles) against on-chip generated keys (black histogram). Externally injected Endorsement Keys are shown in red.

the large numbers [SNS+16] works with. Therefore, we try to detect EK injection using only the most significant byte (MSB) of the modulus of EK(s) compared to the expected distribution of on-chip generated keys. Additionally, we test for the presence of ROCA [NSS+17] fingerprint. We mark an EK as injected if its MSB value was never observed in RSA keys generated on-chip.

Another option is to detect improbable values of MSBs by statistically testing the match between the distribution of MSBs extracted from EKs and MSBs observed in on-chip generated keys. However, such an option requires a reasonably high number of Endorsement Keys (at least higher tens) to have high confidence in the result obtained. As we typically do not possess such a number of EKs (with the exception of IFX 5.63.13.6400 and NTC 7.2.2.0), we do not use this method.

### 5.2.1 Externally injected Endorsement Keys

For AMD and Intel firmware TPMs, we detected no impossible MSBs indicating injected keys. As the AMD and Intel fTPMs are not certified under CC or FIPS 140, no information about the EK generation process is available. However, due to the high overall performance of CPU-based fTPMs, the RSA key generation is expected to be fast enough despite the prime-related variance, removing the need for external generation. Detection of injected keys in fTPMs would be of potential concern, but we found no keys indicating such a case.

On the contrary, we detected externally generated keys for almost all the physical dTPMs. Impossible MSBs or missing ROCA fingerprints for ROCA-vulnerable implementation were found for two out of three ranges of observed RSA implementations for Infineon TPMs. Although the only single EK value available to us for the range IFX 7.83-85.x is technically not impossible (as this implementation sometimes generates all MSB values), it is very improbable with less than 0.1% of on-chip generated keys having such MSB.

For IFX 5.63.13.6400 with a total of 225 EKs available, we additionally computed a histogram (bin size = 10) of observed MSBs, demonstrating a significantly different distribution for the source of injected EKs. While the certification document for this TPM does not identify the exact HSM used[17], the distribution matches Utimaco Security Server Se50 as shown by [JNS+20] – which might be the HSM used by Infineon.

Impossible MSB was detected for the Nuvoton 7.2.2.0 version. We detected no impossible MSBs for TPMs manufactured by STM despite the corresponding certificates explicitly describing EK generation with FIPS 140-2 compliant HSM. The reason is likely due to the low number of EKs available to us (only 7), possibly also due to the usage of HSM with a similar distribution of MSBs as the on-chip generation.

Similarly to Endorsement Keys, we tested Storage Root Keys' properties for any signs of deviation from the distribution expected for on-chip generated keys. The Storage Root Key (SRK) is a long-term key regenerated only when a user takes ownership of the module (operation typically performed only sporadically) and stored on a chip. SRK must be generated by the TPM chip itself, as the end user can repeatedly take ownership. SRK's properties may still be different (e.g., backdoored implementation would have a different way to generate SRKs, which are regenerated only very infrequently) but are expected to be the same as a chip's common ones. As shown in Figure 5, we detected no deviation from the expected SRK distribution.

Overall, we can conclude that Endorsement Key injection was detected where expected for dTPMs based on their certification claims and *not* detected for fTPMs and iTPMs where it is functionally viable to generate such key directly on-chip. No suspicious deviance was detected for any Storage Root Key.

# 6 Algorithms support and performance

The main TCG TPM specification [TCG19b] defines a number of algorithms, elliptic curves, and commands, not all of which need to be supported on all platforms, and thus their support varies among different TPM implementations. Platform-specific requirements for personal computers are defined by the TCG PC Client Platform TPM Profile (PTP) specification [TCG20]. In this section, we present how frequent is the support of particular algorithms, elliptic curves, and selected commands in TPM firmware we encountered. The dataset contains this information only for measurements performed using the `tpm2-algtest` tool, which used the `tpm2_getcap` command from `tpm2-tools`[18] with options `algorithms`, `ecc-curves`, `commands` to obtain the support information.

The observations are discussed below, with percentage support for selected algorithms summarized in Table 2, complemented by the information on whether the support of a particular algorithm, elliptic curve, or command is mandatory or only optional according to the latest PTP [TCG20].

**Algorithms** The common cryptographic algorithms, including RSA cryptosystem, SHA1 and SHA256 hash functions, AES cipher, and HMAC, were supported by all TPM firmware

[17] https://seccerts.org/cc/1d1df0fb541e49b8/target.pdf.

[18] https://github.com/tpm2-software/tpm2-tools.

**Table 2:** Observed percentage support of TPM algorithms by firmware in the dataset. Each firmware sample was counted only once. M (mandatory), O (optional), M- (used to be mandatory but is not anymore), M+ (used to be optional but now is mandatory), M? (used to be mandatory but now is not mentioned), – (not mentioned).

| TPM2_ALG_* | PTP | % |
|---|---|---|
| *ECC-related* | | |
| ECC | M | 98 |
| ECDH | M | 98 |
| ECDSA | M | 98 |
| ECDAA | M- | 89 |
| ECSCHNORR | M- | 72 |
| ECMQV | O | 0 |
| SM2 | O | 12 |
| *RSA-related* | | |
| RSA | M | 100 |
| RSASSA | M | 100 |
| RSAES | M | 100 |
| RSAPSS | M | 100 |
| OAEP | M | 100 |
| *Hash functions* | | |
| SHA1 | M | 100 |
| SHA256 | M | 100 |
| SHA384 | M+ | 28 |
| SHA512 | O | 0 |
| SHA3_256 | O | 5 |
| SHA3_384 | O | 5 |
| SHA3_512 | O | 0 |
| SM3_256 | O | 11 |

| TPM2_ALG_* | PTP | % |
|---|---|---|
| *Block ciphers* | | |
| TDES | O | 0 |
| AES | M | 100 |
| CAMELLIA | O | 0 |
| SM4 | O | 11 |
| *Encryption modes* | | |
| ECB | – | 70 |
| CBC | – | 70 |
| CTR | – | 84 |
| OFB | – | 84 |
| CFB | – | 100 |
| *Other* | | |
| XOR | M | 100 |
| HMAC | M | 100 |
| CMAC | – | 0 |
| MGF1 | M | 74 |
| KEYEDHASH | M | 100 |
| SYMCIPHER | M | 100 |
| KDF1_SP800_56A | M? | 87 |
| KDF1_SP800_108 | M? | 100 |
| KDF2 | – | 0 |

| TPM2_ECC_* | PTP | % |
|---|---|---|
| NIST_P192 | – | 0 |
| NIST_P224 | – | 7 |
| NIST_P256 | M | 98 |
| NIST_P384 | M+ | 41 |
| NIST_P521 | – | 0 |
| BN_P256 | M- | 87 |
| BN_P638 | – | 0 |
| SM2_P256 | – | 11 |

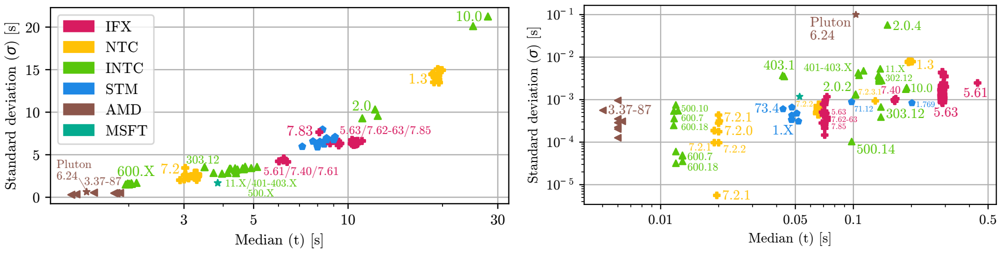
| TPM2_CC_* | PTP | % |
|---|---|---|
| ECDH_KeyGen | M | 98 |
| ECDH_ZGen | M | 98 |
| EC_Ephemeral | O | 82 |
| ZGen_2Phase | O | 82 |
| Commit | M- | 97 |
| Sign | M | 100 |

we encountered. Similarly, common ECC algorithms like ECDH and ECDSA on the NIST_P256 curve were supported by all but one old Intel fTPM (INTC 9.5.65.3000).

The support of the ECDAA algorithm is usually related to the support of BN_P256 curve and the Commit command, which are supposed to be used jointly, and they were absent only on Intel fTPMs that comply with the TPM revision from the year 2013 or earlier. Nuvoton TPMs have stopped supporting both the ECDAA algorithm and the BN_P256 curve with the version 7.2.3.1. Surprisingly, the MSFT 6.3.1.603 firmware seems to be supporting the ECDAA algorithm but not the BN_P256 curve. Algorithms from the SM specifications were supported only by Intel firmware from the latest 600.x series. The Schnorr signature algorithm ECSCHNORR used to be mandatory by the PTP, but it is not anymore since the last revision v1.59 [TCG20]. It was not supported by Intel fTPMs complying with the 2013 specification or earlier and any of the Infineon TPMs. Nuvoton TPMs used to support this algorithm, but removed the support with version 7.2.3.1.

Hash algorithms from the SHA3 family were supported only by STM firmware 1.258, 1.512, and 1.769. Hash algorithms with 384-bit output were also supported by these STMs, but additionally also by Intel 600.x series and 7.2.x.x Nuvoton TPMs. No TPM supported hash algorithms with 512-bit output. All TPMs supported CFB encryption mode. No other mode was supported by Infineon TPMs, and ECB and CBC were also not supported by Nuvoton TPMs.

**Elliptic curves** The support of at least NIST_P256 curve has been mandatory for all TPMs for the PC platform [TCG20], and this is confirmed by our observations, where the only TPM not supporting this curve is INTC 9.5.65.3000, predating the PTP specification and not supporting ECC at all. The second most supported curve present on all TPMs



Variability of RSA 2048b key generation

Variability of ECDSA-P256 signature

**Figure 6:** An observed variance of RSA-2048b key pair generation and ECDSA-P256 sign operations with median time and standard deviation is shown over all collected firmware versions (note the logarithmic axes). Firmware clusters and notable outliers are annotated.

complying with a specification revision from the year 2014 or later is the pairing-friendly `BN_P256`; support of this (or an alternative pairing-friendly) curve is required to enable the `ECDAA` protocol. The support of curves with lower bit-length than 256 bits is quite rare and has been observed only on older STM TPMs. The curves with a larger bit-length are more common, which is partly caused by `NIST_P384` being mandatory in later revisions of the PTP specification [TCG20]. The Chinese `SM2_P256` curve has been supported only by 600.x series Intel firmware.

**Commands** The basic building blocks of advanced applications utilizing TPMs rely on six commands, as summarized by Chen and Urian [CU16]; the support for these operations among TPM firmware we analyzed is listed in the last section of Table 2. The majority of these commands are commonly supported by current TPMs; only `EC_Ephemeral` and `ZGen_2Phase` are not supported by Infineon TPMs, and `Commit` by the latest NTC 7.2.3.1.

Asymmetric decryption schemes [CU16] seem computable on all recent TPMs as they require only support for `ECDH_KeyGen` or `ECDH_ZGen`. Anonymous attestation and multi-party protocols utilize the two-stage construction based on `Commit` and `Sign` together with the `ECDAA` algorithm and, in some cases, also pairing friendly `BN_P256` curve. Their joint support is quite common, with the only exception being the latest NTC and MSFT firmware.

The lowest practical rate of support is for the two-stage key-agreement protocols, not only because Infineon TPMs do not support these commands at all but also because they require the support of other algorithms to be fully usable. Apart from the basic `ECDH`, the `SM2` support is present only in 600.x series Intel TPMs and `ECMQV` support has not been found in any of the firmware.

## 6.1 Performance of operations

We collected and analyzed performance for all supported algorithms using repeated calls of the corresponding method using `tpm2-algtest` software. Each method invocation was provided with the same fixed input data and was repeatedly measured 1000 times using a high-precision host clock. However, some internal randomization may influence the resulting time, like the generation of RSA/ECC keys, generation of random data from stochastic processes, or execution of protected implementations. If a TPM firmware version is measured multiple times (more machines with the same TPM available), we verify whether the observed properties persist.

Due to the overall large number of firmware versions analyzed, we present only aggregated results for two operations analyzed in Sections 4 and 5 – `TPM2_Create(RSA-2048b)` and `TPM2_Sign(ECDSA-P256)`, capturing the median and standard deviation of all measurements from physical chips with a given firmware version. The results are shown in Figure 6 with notable firmware clusters annotated, providing several insights.

All clusters observed for the generation of RSA-2048b keypairs directly map to change in the prime generation algorithm analyzed in Section 5.1, providing additional evidence of change in the cryptographic implementation. The only exception is a separation of Intel 500.x (Jasper Lake) and 600.x (Alder and Raptor Lake) versions where faster generation for 600.x can be attributed either to faster underlying CPU architecture or other (unknown) changes to RSA keypair generation. The clusters observed for ECDSA-P256 signatures are more complicated to analyze, partly due to significantly shorter operation time, resulting in a higher impact of measurement noise and, thus, more overlapped clusters. Median time is faster with the later firmware revisions on Intel's fTPMs, with one notable exception – version 500.14.x (Comet Lake), which is almost 10x slower than (numerically) preceding version 500.10.x (Jasper Lake). Pluton-based iTPMs are discussed in detail in the next section. While the timing difference can be again explained by a change in the underlying CPU architecture, differences observed also illustrate the difficulty in assessing the impact of a vulnerability based simply on numerical version ordering if a vendor does not consistently report all the vulnerable versions.

## 6.2 Properties of Pluton-based iTPMs

Pluton is a recent secure crypto-processor[19] designed by Microsoft together with CPU vendors (AMD, Intel, Qualcomm) responsible for the chip production. The first listed CPUs with Pluton support are Ryzen 6000 and Qualcomm Snapdragon 8cx Gen 3 series.

The Pluton chip can be utilized either as a special Windows-only Pluton device or as an integrated TPM (iTPM) with custom cryptographic hardware (in contrast to fTPMs), directly embedded inside the main CPU (in contrast to dTPMs), thus limiting the accessibility of the communication bus for malicious sniffing.

As only a little public information about the design is available and no certificates under Common Criteria or FIPS140-3 have been issued yet[20], we present one of the first cryptographic results for the Pluton chip and its firmware, using its observable cryptographic properties.

We obtained two Pluton-based chips for the analysis: AMD Ryzen 9 7950x (launch date September 2022, TPM firmware version 'AMD 6.24.0.6', rev. 1.59) and AMD Ryzen 7 PRO 7840U (launch date June 2023, TPM firmware version 'MSFT Pluton.TPM.A 6.3.1.603', rev. 1.38), both used as TPMs. Despite the seemingly similar firmware numbering of AMD 6.24.0.6 and MSFT 6.3.1.603, the performance and behavioral characteristics are different enough to likely constitute entirely different implementations.

The cryptographic characteristics (RSA, ECC) of iTPM AMD 6.24.0.6 (Ryzen 9 7950x) are observably the same as for all previous versions of AMD's fTPMs, just with a significantly slower performance than its predecessors for most operations. The fastest operation (`TPM2_GetRandom`) requires around 135 ms to execute in comparison to less than 3 ms on the previous AMD fTPMs. Such behavior would be consistent with the reuse and adaptation of AMD's existing fTPM source code for cryptographic operations like RSA key and ECC nonce generation but executed using a slower physical Pluton chip (iTPM) instead of a main CPU (fTPM). The data transfers between the main CPU and Pluton chip and different key object handling may be responsible for the significant slowdown. Additionally, we observed linearly increasing timing of all operations generating

[19] https://learn.microsoft.com/en-us/windows/security/hardware-security/pluton/microsoft-pluton-security-processor.

[20] *Microsoft Pluton Security Processor ROM* for AMD Ryzen 6000 series, which is currently under review process of FIPS140-3 with status On Hold (12/11/2023) https://csrc.nist.gov/Projects/cryptographic-module-validation-program/modules-in-process/Modules-In-Process-List.

new symmetric (AES, HMAC) or asymmetric (ECC) keys. A key generation runs a bit slower after every key generation, resulting cumulatively in ~14 ms slowdown after 1000 previous invocations, possibly caused by internally growing keys storage structure or bad memory management. No such slowdown was observed for the MSFT implementation.

Microsoft's implementation of iTPM firmware version MSFT 6.3.1.603 is observably different from AMD 6.24.0.6. The TPM revision is still 1.38, and the set of supported algorithms is smaller, with no support for RSA-3072b and curve `TPM2_ECC_BN_P256`, despite the underlying CPU chip being released almost one year later. Additionally, the execution times for almost all operations are an order of magnitude faster on MSFT 6.3.1.603 than on AMD 6.24.0.6 (e.g., median 26 ms vs. 210 ms for AES encrypt/decrypt), with two notable exceptions. Firstly, the signature time for RSA-2048b (both `TPM2_ALG_RSASSA` and `TPM2_ALG_RSAPSS`) is practically identical for MSFT and AMD iTPMs, while the same operation with RSA-1024b is more than 4x *faster* on MSFT (median 51 ms vs. 207 ms). Either the underlying Pluton chip in the desktop AMD CPU is significantly faster, or Microsoft's implementation for laptop CPUs is better optimized (possibly at data transfer) to utilize the Pluton chip. Given almost the same timing of the RSA-2048b signature operation, the latter case is more probable. Secondly, the RSA key generation by MSFT implementation is 5x *slower* for RSA-1024b and more than 2x *slower* for RSA-2048b than AMD, hinting at very different implementations of a prime generation algorithm. The distribution of MSBs of primes observed for MSFT is the same as used by Microsoft for its software-only cryptographic libraries CryptoAPI/CNG/.NET (as documented by [SNS$^+$16]), but without $p \pm 1$ factors having 101-120 bit length only[21].

Surprisingly, despite the Pluton-based AMD 6.24.0.6 being at least an order of magnitude slower than all older firmware-only AMD fTPMs in almost all operations, the RSA key generation of the Pluton-based AMD is very fast – with the median speed of RSA-2048b keygen even faster than several older AMD fTPMs. At the same time, MSFT's implementation, also using the Pluton chip, is several times slower for this operation (median 2129 ms vs. 414 ms), with a performance on par with dTPMs like Infineon or Nuvoton. Additionally, RSA key generation is the only key generation operation where the slowdown was not observed for AMD, with an increasing number of keys generated (as discussed above). We may hypothesize that the AMD 6.24.0.6 performs the RSA key generation *outside* the Pluton chip (same as previously done for the AMD fTPMs). Such implementation choice would result, apart from better performance, in potentially lower security guarantees as a private key value may be exposed during key generation to a less secure generic CPU environment. Alternatively (but unlikely), AMD Pluton-based implementation might be several times more efficient than MSFT in its utilization of the Pluton chip but just for the RSA key generation while significantly slower for all others.

To summarize, the Pluton-based iTPMs (at least the two initial ones based on the Ryzen 6000 series) are cryptographically similar to their fTPM predecessors (possibly being only a port of an existing source code) and performance-wise visibly different from dTPMs and fTPMs. Generally slower than fTPMs and faster or equal to dTPMs, Pluton-based implementation additionally varies significantly with the particular firmware implementation (AMD or MSFT). While for the dTPMs, all cryptographic operations are verifiably (e.g., by monitoring data bus) executed inside the discrete chip, such assurance is not readily available for Pluton-based iTPMs embedded inside a CPU. A vendor may be tempted to implement some operations outside the Pluton chip (a possible example being RSA key generation for iTPM AMD 6.24.0.6) for performance or other reasons, resulting in possibly lower security guarantees, yet such change would be difficult to detect using black-box-only analysis.

[21] Strong primes, as required by ANSI X9.31 to slow down certain factorization methods.

# 7 Conclusions

By performing a comprehensive study on 78 different TPM firmware versions, we identified relatively frequent changes in cryptographic code with an observable impact on cryptographic keys. Additionally, we detected numerous cases of unreported or inconsistently documented vulnerabilities. The original researchers frequently missed the same or similar weaknesses in other cryptographic algorithms or firmware revisions, which is understandable given the black-box nature of these TPM implementations and the necessity to possess the particular TPM firmware version for testing. More worryingly, the TPM vendors frequently failed to report all vulnerable TPM versions in their security bulletins addressing the reported vulnerability and did not fix all the occurrences of the same vulnerability type. Failure to do so makes it particularly difficult for end-users to assess if they were impacted and shall update firmware or even consider their cryptographic keys fully compromised. Nuvoton did not report TPM-Fail-like leakage present also for ECDAA and ECSCHNORR algorithms of the version named in the TPM-Fail disclosure and earlier firmware versions. While Intel reported on the whole range of TPM-Fail-affected firmware versions, no public disclosure was made about another vulnerability present and patched in a subset of these versions. That vulnerability (newly reported in this paper) requires only nine signatures, without any side-channel measurement, to extract the used private key – a strong attack that can be mounted even retrospectively. Intel either fixed it unintentionally while moving to the next firmware version with more extensive changes (also visible in RSA key generation) or silently with no disclosure.

The availability of a wide range of firmware versions allows us to evaluate the level of support and performance of cryptographic algorithms – especially ones necessary for more advanced protocols like anonymous attestation or multiparty signatures.

Finally, a first black box analysis of cryptographic and performance properties of a recently introduced Microsoft Pluton cryptographic co-processor with AMD and Microsoft firmware documents significant implementation differences and hints at RSA key generation possibly occurring outside the Pluton chip for the AMD iTPM firmware.

## Acknowledgements

P. Svenda, A. Dufka, R. Lacko and T. Jaros were supported by Ai-SecTools (VJ02010010) project. The data collection was in part supported by a grant from the Cisco University Research Program Fund, Silicon Valley Community Foundation.

## References

[BEB18] Robert G Brown, Dirk Eddelbuettel, and David Bauer. Dieharder. *Duke University Physics Department Durham, NC*, pages 27708–0305, 2018.

[BGG10] Bernhard Beckert, Daniel Grahl, and Sarah Grebing. Mind the gap: Formal verification and the Common Criteria. In *VERIFY@IJCAR*, 2010.

[BRS+10] Lawrence Bassham, Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Stefan Leigh, M Levenson, M Vangel, Nathanael Heckert, and D Banks. A statistical test suite for random and pseudorandom number generators for cryptographic applications, 2010-09-16 2010.

[BV96] Dan Boneh and Ramarathnam Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In *Annual International Cryptology Conference*, pages 129–142. Springer, 1996.

[CL13] Liqun Chen and Jiangtao Li. Flexible and scalable digital signatures in TPM 2.0. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 37–48, 2013.

[CU15] Liqun Chen and Rainer Urian. DAA-A: Direct anonymous attestation with attributes. In *8th International Conference on Trust and Trustworthy Computing, Proceedings 8*, pages 228–245. Springer, 2015.

[CU16] Liqun Chen and Rainer Urian. Algorithm agility – discussion on TPM 2.0 ECC functionalities. In *Security Standardisation Research: Third International Conference, Proceedings 3*, pages 141–159. Springer, 2016.

[HA20] Mustapha Hedabou and Yunusa Simpa Abdulsalam. Efficient and secure implementation of BLS multisignature scheme on TPM. In *IEEE Conference on Intelligence and Security Informatics (ISI'20)*, pages 1–6. IEEE, 2020.

[Hea04] J. Hearn. Does the Common Criteria paradigm have a future? *IEEE Security & Privacy Magazine*, 2(1):64–65, 2004.

[HTAP18] Julie Haney, Mary Theofanos, Yasemin Acar, and Sandra Spickard Prettyman. Organizational views of NIST cryptographic standards and testing and validation programs. Technical Report NIST IR 8241, NIST, December 2018.

[Jan23] Jakub Janku. Schnorr multi-signatures for secure devices with restricted interfaces, 2023. Bachelor's thesis, Masaryk University, https://is.muni.cz/th/p8u2x/.

[Jer18] Jeremy Boone. TPM Genie: Interposer attacks against the trusted platform module serial bus. https://github.com/nccgroup/TPMGenie/blob/master/docs/NCC_Group_Jeremy_Boone_TPM_Genie_Whitepaper.pdf, 2018.

[JJS+23] Adam Janovsky, Jan Jancar, Petr Svenda, Łukasz Chmielewski, Jiri Michalik, and Vashek Matyas. sec-certs: Examining the security certification practice for better vulnerability mitigation. *arXiv:2311.17603*, 2023.

[JNS+20] Adam Janovsky, Matus Nemec, Petr Svenda, Peter Sekan, and Vashek Matyas. Biased RSA private keys: Origin attribution of GCD-factorable keys. In *ESORICS'20*. Springer, 2020.

[JSSS20] Jan Jancar, Vladimir Sedlacek, Petr Svenda, and Marek Sýs. Minerva: The curse of ECDSA nonces; systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):281–308, 2020.

[JWBS23] Hans Niklas Jacob, Christian Werling, Robert Buhren, and Jean-Pierre Seifert. faulTPM: Exposing AMD fTPMs' deepest secrets. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroSP'23)*, pages 1128–1142, 2023.

[KBR14] Samuel Paul Kaluvuri, Michele Bezzi, and Yves Roudier. A quantitative analysis of Common Criteria certification practice. In *TrustBus 2014*, volume 8647 of *LNCS*, pages 132–143. Springer, 2014.

[KK19] Yongjin Kim and Evan Kim. HTPM: Hybrid implementation of trusted platform module. In *Proceedings of the 1st ACM Workshop on Workshop on Cyber-Security Arms Race*, CYSARM'19, page 3–10, New York, NY, USA, 2019.

[LLL82] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261(ARTICLE):515–534, 1982.

[LS07] Pierre L'ecuyer and Richard Simard. TestU01: AC library for empirical testing of random number generators. *ACM Transactions on Mathematical Software (TOMS)*, 33(4):1–40, 2007.

[MBA12] Steven Murdoch, Mike Bond, and Ross J. Anderson. How Certification Systems Fail: Lessons from the Ware Report. *IEEE Security & Privacy Magazine*, pages 1–1, 2012.

[MSEH20] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, and Nadia Heninger. TPM-FAIL: TPM meets timing and lattice attacks. In *USENIX Security 2020*, pages 2057–2073. USENIX Association, 2020.

[NSS+17] Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, and Vashek Matyas. The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli. In *ACM SIGSAC CCS'17*, pages 1631–1648. ACM, 2017.

[PZ11] Christian Paquin and Greg Zaverucha. U-Prove cryptographic specification v1.1. *Technical Report, Microsoft Corporation*, 2011.

[SETA21] Chao Sun, Thomas Espitau, Mehdi Tibouchi, and Masayuki Abe. Guessing bits: Improved lattice attacks on (EC)DSA with nonce leakage. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):391–413, Nov. 2021.

[SKKS19] Marek Sys, Dusan Klinec, Karel Kubicek, and Petr Svenda. Booltest: The fast randomness testing strategy based on boolean functions with application to DES, 3-DES, MD5, MD6 and SHA-256. In *E-Business and Telecommunications: ICETE 2017, Revised Selected Paper 14*, pages 123–149. Springer, 2019.

[SNS+16] Petr Svenda, Matus Nemec, Peter Sekan, Rudolf Kvasnovsky, David Formanek, David Komarek, and Vashek Matyas. The Million-Key Question – Investigating the Origins of RSA Public Keys. In *The 25th USENIX Security Symposium (UsenixSec'2016)*, pages 893–910. USENIX, 2016.

[TCG09] TCG. ISO/IEC 11889-1:2009. https://www.iso.org/standard/50970.html, 2009.

[TCG17] TCG. TCG TPM v2.0 provisioning guidance. https://trustedcomputinggroup.org/wp-content/uploads/TCG-TPM-v2.0-Provisioning-Guidance-Published-v1r1.pdf, 2017.

[TCG19a] TCG. TPM 2.0 specification, section 14.3.5. https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf, 2019.

[TCG19b] TCG. Trusted Platform Module Library Specification, Family "2.0", Level 00, Revision 01.59, 2019.

[TCG20] TCG. PC client platform TPM profile (PTP) specification. https://trustedcomputinggroup.org/resource/pc-client-platform-tpm-profile-ptp-specification/, 2020.

[TCG23a] TCG. TCG certification programs. https://trustedcomputinggroup.org/membership/certification/, 2023.

[TCG23b] TCG. TPM Certified Products. https://trustedcomputinggroup.org/membership/certification/tpm-certified-products/, 2023.

[TCP01] TCPA. Trusted Computing Platform Specifications v1.0. https://web.archive.org/web/20020806140425/http://www.trustedcomputing.org/docs/tcpa_final.pdf, 2001.