Partial Key Overwrite Attacks in Microcontrollers: a Survey

Abstract. Embedded devices can be exposed to a wide range of attacks. Some classes of attacks can be mitigated using security features or dedicated countermeasures. Examples include Trusted Execution Environments, and masking countermeasures against physical side-channel attacks. However, a system that incorporates such secure components is not automatically a secure system. Partial Key Overwrite attacks are one class of attacks that specifically target the interface between different components of the security system. These attacks may allow an adversary to extract otherwise protected cryptographic keys through careful manipulation of memory-mapped registers. So far this powerful class of attacks has received little attention in the academic literature. In this work, we provide an overview of known Partial Key Overwrite vulnerabilities and how they were used in real-world attacks. Additionally, we evaluated 31 common microcontrollers and embedded microprocessors from eleven distinct vendors and detail our findings. Based on a first high-level evaluation we selected 15 SoCs and performed an in-depth evaluation. This evaluation revealed that at least eight of these SoCs are vulnerable to partial key overwrite attacks.

Keywords: key overwrite attack · safe error analysis · microcontrollers · embedded security

1 Introduction

Microcontrollers and microprocessors are used in many embedded devices and deployed in a wide range of applications, from consumer IoT to industrial and automotive. As a result, a single microcontroller must be designed to withstand diverse threats depending on the context. This varies from attackers targetting software vulnerabilities in the firmware, to active and passive semi-invasive hardware attacks. Countermeasures against such attacks range from hardware features in the CPU to enhance software security such as Intellectual Property Encapsulation (IPE) and Trusted Execution Environments (TEEs) to defences in cryptographic accelerators against side-channel analysis and differential fault analysis attacks.

However, securing every individual component separately does not necessarily mean the entire system is secure. One overlooked attack that arises from a naive composition of individually secured components is the Partial Key Overwrite (PKO) attack: it allows an attacker to obtain a cryptographic key otherwise stored in a secure location, typically by careful manipulation of the memory-mapped input/output (MMIO) register interface.

While the concept of PKO attacks has been around for a few decades, they have received little academic attention. The aim of this paper is twofold: raise awareness of the existence of these attacks by listing a few real-world cases, and survey whether common microcontrollers are vulnerable to such attacks or if vendors incorporate any countermeasures against these attacks. We also provide the source code used for checking whether the microcontrollers and SoCs we tested are vulnerable.

1.1 Responsible Disclosure

For every microcontroller in which we discovered a vulnerability, the vendor has released a newer chip in which the vulnerability was mitigated. Hence, a full Coordinated Vulnerability Disclosure procedure was not deemed necessary, although we still notified all chip vendors with confirmed in-scope and vulnerable products of our upcoming publication.

Additionally, we found a peculiar strangeness in the behaviour of the cryptographic accelerator in the Renesas RA2E1 series of microcontrollers. Although its security implications are unclear, it was nonetheless disclosed to Renesas.

1.2 Structure of This Paper

This paper starts with background information in Section 2 and a theoretical description of partial key overwrite attacks in Section 3. Section 4 details a survey investigating which microcontrollers are vulnerable to PKO attacks. Finally, Section 5 provides possible countermeasures and Section 6 makes a few final notes.

2 Background and Related Work

A few publications exist that describe PKO attacks in OpenPGP [14, 7] and the online file storage service MEGA [3, 2]. OpenPGP private keys are stored at rest in an encrypted form, the user has to enter a password to decrypt and use the private key. Similarly, the MEGA service provider (which, in their threat model, should not be trusted by the user) stores the encrypted key material which is only decrypted client-side. However, in either case, parts of the key file can be corrupted, such as elliptic curve parameters stored in plaintext, or a single block of an ECB-encrypted private key. Intercepting a signature made with such a corrupted key allows an attacker to then derive the private key. With such a private key, they can decrypt the files of the user it belongs to.

The above attacks exist in a context rather different from embedded security. Examples of PKO attacks in the latter context are difficult to find in existing literature, though a few can be found. For instance, while [4] is best known as the seminal paper introducing Differential Fault Analysis, in its Section 3 the authors present a different attack, making use of partial key erasure. Their example presents a discrete encryption device with a built-in key stored in an

EEPROM. An attacker sets a few key bits to 1 by exposing the target to ionising radiation. Similarly, the authors of [13] used the zeroisation of S-boxes as a method for key recovery, albeit in a white-box context (that is, they overwrite the part of a software program that implements the S-box). Almost a decade later, a similar attack was performed on an FPGA bitstream [28].

Meanwhile, hacking communities have been using partial key overwrite attacks against many targets. Informal sources describe its use on AES coprocessors in respectively the Nintendo DSi [15], Nvidia Tegra X1 [29] (used in the Nintendo Switch), and STM32H730 [12]; while [1] uses it on an RSA coprocessor in the Nintendo 3DS. In each of these cases, the system initialized the key registers of their coprocessors early during boot, with keys stored in e.g. one-time-programmable (OTP) memory. This memory is then made unreadable, after which the system executes lower-privilege code the attackers are able to exploit. Such an exploit is unable to read the keys from OTP memory or dump the system bootcode, but using a PKO attack, the keys can still be extracted from their accelerators. An attacker does not need a software exploit in the typical sense as a requirement for the PKO attack, in [12] direct debug access is used instead. The debugger cannot access flash memory (STM32 readout protection level 1 is used) but can still access peripheral registers. Finally, it is important to note that attackers have an explicit interest in recovering the key, rather than merely using the accelerator as a potential decryption oracle [16, 9]. Game consoles typically encrypt and sign both the entire filesystem as well as individual games and software binaries. Installing unauthorized software on such a device thus requires circumventing every protection layer. Performing the encryption and decryption of full game install images — or even an entire filesystem — offline instead of having to query the console itself for this is much less of a hassle.

Table 1 gives a summary of all previously-published attacks, along with which cipher they targetted and which method was used to mount the attack.

Target	**Cipher**	**Key storage**	**Method**	**PKO?**	**Circle**	**Citation**
OpenPGP	RSA, ECC	file	client-side malware	✔	academia	[14, 7]
MEGA	RSA, ECC	file	misbehaved server	✔	academia	[3, 2]
(vague)	generic	EEPROM	UV light	✔	academia	[4]
software	AES, DES	(vague)	software tampering	✘ (S-box)	academia	[13]
FPGA	AES, DES	bitstream	bitstream tampering	✘ (S-box)	academia	[28]
(vague)	RSA	(vague)	FI (SEA)	✔	academia	[33]
(vague)	DES	(vague)	FI (IFA)	✔	academia	[8]
Nintendo DSi	AES	WO MMIO reg	VRISKA (exploit)	✔	hobbyist	[15]

Nintendo 3DS	RSA	WO MMIO reg	VRISKA (exploit)	✔	hobbyist	[1]
Tegra X1	AES	WO MMIO reg	VRISKA (exploit)	✔	hobbyist	[29]
STM32H730	AES	WO MMIO reg	VRISKA (debug pins)	✔	hobbyist	[12]
Many	AES, RSA	WO MMIO reg	VRISKA	✔	academia	you are here

Table 1: Summary of published attacks making use of Partial Key Overwriting and similar techniques. The rows list the target attacked, the ciphers involved, the type of key storage used ('WO' means 'write-only'), the method used to mount the attack, whether the attack makes use of partial key overwrites, which community the attack was discovered in, and the relevant citation(s).

2.1 Attack Categorisation

There is no real agreement to which class of attacks PKO attacks belong. The authors of [34, Section VI.B.3] mention PKO as a subtype of Safe Error Analysis (SEA) attacks[1]. However, SEA is more widely understood as a subclass of (physical) fault injection (FI) attacks similar to Ineffective Fault Analysis (IFA) [33, 8, 34]. Hence, referring to PKO attacks as SEA might lead to confusion.

While referring to PKO as SEA makes sense from a purely cryptographic point of view, this paper focuses on PKO attacks in embedded devices, where they arise from imperfections in the interface between components. This is a property they have in common with Interrupt Oriented Programming [30], DMA-based attacks such as [6], the use of 'hardware gadgets' in [22], the `ntrcardhax` exploit in the Nintendo 3DS[2] [25] and the vulnerability in the Falcon TSEC coprocessor in the Tegra X1 [24].

As far as we are aware, no properly defined category for the attacks in the previous paragraph seems to exist. Yet, these attacks make use of vulnerabilities that have several elements in common: The vulnerabilities lie not in a single component of the system-on-chip (such as the CPU, memory, an accelerator, etc.), but rather arise from the combination of these components in a system, often due to complexities in the interface between them (such as their register interface, DMA, interrupts, 'event system/routing', etc.). We hence propose to name the category of attacks that exploit these vulnerabilities Vulnerable Result of Individually Secure KComponents Attacks, or *VRISKA*s[3] for short.

We can view the PKO attacks treated in this paper as SEA mounted using a VRISKA. This differs from the attacks on OpenPGP and MEGA which are mounted using software vulnerabilities, and from 'traditional' SEA which

[1] "write-only cryptographic key registers should never allow partial update, otherwise the attacker can test a partial key guess by detecting these collisions." [2] Not to be confused with `ntrboot`, a *different* exploit in the 3DS system. [3] Yes, after the *Homestuck* character *Vriska Serket*.

is mounted using FI. Note that while more 'traditional' SEA attacks use transient faults, the modifications made to the key register in this work are more permanent in nature: they last until the next reset of the chip.

Similarly, with the above definition, `ntrcardhax` from [25] used a VRISKA on a time-of-check-vs-time-of-use vulnerability to perform a buffer overflow, [24] builds a ROP chain using a VRISKA, and [22] uses one as a source of side-channel information.

3 Theory of Partial Key Overwrite Attacks

In this section, we first define the attacker model. Afterwards, we describe how PKO attacks may be used against cryptographic coprocessors.

3.1 Attacker Model

In a PKO attack on a cryptographic coprocessor, the attacker can query the coprocessor as an encryption oracle and is able to overwrite parts of the key used. This key is never directly exposed to the attacker, it is often made inaccessible through some sort of privilege separation mechanism. It is the goal of the attacker to obtain the key.

Typically, the attacker takes control of the CPU to query the cryptographic coprocessor, but other possibilities exist as well, such as debug access. Similarly, the 'privilege separation mechanism' can come in many forms. Sometimes, the key is loaded into the coprocessor by firmware code running inside a TEE or otherwise locked away by an early system boot stage. In other cases, it might be loaded from some protected memory region such as one-time-programmable memory. In the case of debug access instead of software exploits, special code readout protection mechanisms serve as this privilege separation.

3.2 Attack Method for Symmetric-Key Coprocessors

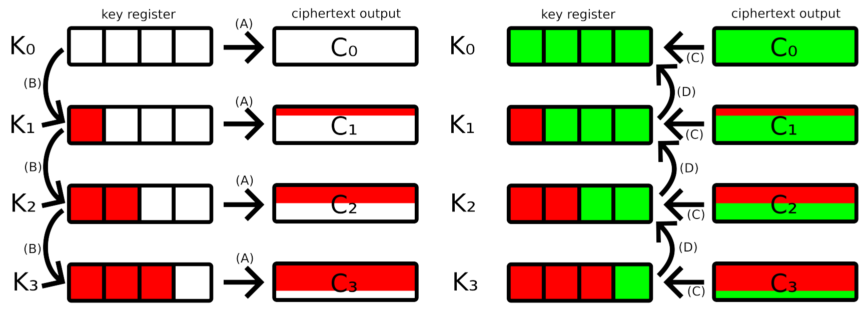
The attack consists of two phases: obtaining ciphertexts with partly-overwritten keys, and a 'brute-force' calculation to obtain the key. These are depicted in Figure 1.

In the first phase, the attacker will start by querying the encryption oracle with its original key K_0 using a known plaintext P, receiving a ciphertext C_0. The exact value of P does not matter, but it must be kept constant across all queries. Then, the attacker proceeds by setting a single subkey[4] to an attacker-chosen value, resulting in the oracle using a key K_1. The attacker then queries the oracle again, obtaining the ciphertext C_1. We define n as the number of subkeys.

[4] For example, an AES-128 coprocessor with a memory-mapped IO (MMIO) interface will have its 128-bit key accessible over this MMIO interface. However, it is likely that this key is larger than the width of the memory bus. Hence, the key is split into multiple subkeys (between 8 and 32 bits in size, typically) which may be written to individually.

This procedure continues until the key is fully overwritten by the attacker save for one word, resulting in a key K_{n-1} with corresponding ciphertext C_{n-1}.

In the second phase, a brute-force calculation with reduced complexity to obtain the original key K_0 is performed. The individual subkeys are sufficiently small to be within reach of brute-forcing (typically 32 bits at most in practice). The attacker can thus work backwards from K_{n-1} to K_{n-2} by knowledge of C_{n-1} and C_{n-2}, after which K_{n-3} is in computational reach, and so on. This continues until K_0 is recovered.



(a) Phase 1: the coprocessor, acting as an encryption oracle, is queried for ciphertexts (A). For each step, the attacker overwrites one more subword of the key (B), while keeping to receive ciphertexts. This ends when all but one subwords of the key are left

(b) Phase 2: in the last query, only one subword K_3 is left. The ciphertext C_3 obtained from the coprocessor makes it possible to brute-force the unknown key bits K_3 (C). This then makes it possible to obtain K_2, and so forth (D), until the full key is derived

Fig. 1: Diagram of a basic PKO attack on a symmetric-key coprocessor

While the above matches the description in [4], variations of this attack are possible as well. For example, the register interface of the coprocessor may expose the key using a shift register instead. This has an influence on the bookkeeping, but the method is otherwise almost unchanged. This contrasts to [15], wherein Korth first queries the oracle with state K_0 to obtain C_0, but then starts writing to the same key register, iterating over every possible value of a subkey, until the same ciphertext C_0 is reached again. This procedure is repeated for every subkey. This attack method thus performs both phases at the same time. Hence, the precise method of attack can be adapted depending on implementation details of the MMIO register interface. Countermeasures have to account for all of these variants.

3.3 Attack Method for RSA Coprocessors

In [1], Myria provided a novel technique for applying PKO attacks to RSA accelerators as well. It assumes the coprocessor has separate registers for message M, private exponent d, and public modulus N.

The attacker sets the message to a small prime M', and the modulus to a prime p greater than N and whose multiplicative order is a smooth number. The exponent register is left untouched. The attacker then queries the oracle, obtaining $C = M'^d \mod p$. With this specific choice of root M' and prime modulus p, it is computationally feasible to calculate $d = \mathrm{dlog}_{M'} C \mod p$ using the Pohlig-Hellman algorithm [21].

No iterative or recursive approach is required, unlike the symmetric-key variant explained in Section 3.2. The attacker must perform some offline computations, however: generating a suitable prime p of which its multiplicative order is a smooth number beforehand, and performing the Pohlig-Hellman algorithm afterwards. This contrasts to the symmetric-key approach, which cay be performed in an entirely online manner.

3.4 Attack Methods for ECC and PQC Coprocessors

It is possible to perform PKO attacks on Elliptic Curve Cryptography (ECC) and Post-Quantum Cryptography (PQC) coprocessors as well. Do note that, while the details of an attack depend on the register interface of the coprocessor, no vulnerable coprocessors implementing either ECC or PQC were found in the survey in Section 4. Hence, the below attacks are merely hypothetical.

Attacks on ECC coprocessors depend on the exact cryptosystem used. For ECDSA, a logical approach for an attacker would be to have a signing oracle generate two different signatures with the same nonce, or at least influence enough of the nonce to perform the lattice attack described in [17]. For Curve25519 and other systems, input parameters could be tweaked to perform an attack similar to the one described in Section 3.3.

For LWE-based PQC schemes such as ML-KEM (also known as Kyber), an attacker could iterate over possible values of every subkey of the secret key reminiscent of the symmetric-key case of [15] in Section 3.2. A decapsulation error would then inform the attacker on whether the subkey guess is correct, reminiscent of [33].

3.5 Practical Considerations

While the calculations required for a PKO attack are in the realm of the practical, these might still take a considerable amount of time on a small microcontroller. This depends on several details, such as the subkey register word size used in the accelerator interface. Performing these attacks efficiently requires a bit of thought.

One may also carry out the second phase described in Section 3.2 in an offline manner. The need for offline attacks depends on the granularity with which the key subwords can be overwritten. If 8-bit writes are possible, a byte-by-byte bruteforce of an AES-128 key requires $16 \cdot 2^8 = 2^{12}$ steps, which is perfectly feasible even on an 8-bit 1 MHz microcontroller, and thus would need no extra offline processing. With a 32-bit write granularity, $4 \cdot 2^{32} = 2^{34} \approx$ 17 billion iterations are instead required. Such a computation takes about 16 minutes single-threaded on an Intel Raptor Lake i7-13700 workstation using AES-NI, while it takes about 6 hours on an STM32L562 at 110 MHz and making use of its AES accelerator.

Additionally, when performing a PKO attack on an RSA coprocessor, the prime modulus p can be chosen by constructing possible candidates of the form $2^x 3^y 5^z + 1$ for any natural numbers x, y and z, and selecting one that is prime and greater than the original RSA modulus. (In our experience, first constructing a number of smooth multiplicative order and then checking for primality yields a usable result much faster than generating a prime and checking for smoothness.) The root (and plaintext) is typically set to 11 or 13. A Python script implementing both this search and the Pohlig-Hellman algorithm needs less than 5 seconds to run either calculation on an Intel Skylake i7-6500U laptop, while performing these computations at all on a microcontroller can be challenging.

4 Prevalence Survey of Vulnerable Cryptographic Coprocessors

In this section, we perform a survey to check how many off-the-shelf microcontrollers and microprocessors contain cryptographic coprocessors vulnerable to PKO attacks as described in the previous section.

4.1 Method

This survey is performed in two parts: in the first part, datasheets and reference manuals of available microcontrollers and microprocessors are scanned, to find possibly-vulnerable candidates. In the second part, these chips are then ordered and some code is loaded onto them that tests for the presence of the vulnerability.

Compiling the list of candidates is done by first compiling a list of common microcontrollers and SoCs with cryptographic accelerators, then selecting chips with some form of software privilege separation, and finally reading their reference manuals to disqualify chips with software-readable key registers. The full list of chips considered (and some not considered) can be found in Table 2. Code used for vulnerability testing can be found at `https://anonymous.4open.science/r/pko-testcode-5D2D`.

4.2 Limitations

The methodology described above skews towards not investigating chips without documentation on their cryptographic coprocessors. For example, for many microcontrollers made by Renesas and large microprocessors/SoCs made by Texas

Instruments, Broadcom or NXP, the documentation on their cryptographic hardware is only available under NDA. Several other microprocessors made by e.g. Qualcomm, Mediatek, Samsung, Amlogic or Rockchip meanwhile have no publicly available documentation at all. Hence, smaller microcontrollers are overrepresented in this work.

Secondly, it would be interesting to know how many *devices* existing *in the wild* contain secrets that may be leaked using PKO attacks. However, performing such a study is much more impractical: it would require 1. buying a large number of off-the-shelf devices, 2. opening them up to discover which microcontroller or microprocessor is used inside (which is often not known before physical disassembly), 3. checking if the chip used has an accelerator vulnerable to a PKO attack, 4. extracting the firmware to check if the vulnerable accelerator is used in a manner in which the vulnerability is actually in scope (the 'privilege separation' explained in Section 3.1 has to be configured properly as well), 5. and finally try to exploit this vulnerability. Such a survey would require buying a large number of devices that might turn out to not use any vulnerable chips at all, while this cost cannot be recovered as the warranty will have to be void (due to disassembly of the device). Hence, this survey only concerns itself with researching which microcontrollers could *possibly* be configured in a way that will lead to key leakage through a PKO attack. We therefore cannot make any statements on the prevalence of PKO vulnerabilities in real-world devices.

4.3 Chip Selection

Most of the chips included in this study store the key in simple write-only MMIO registers (STM32 series, ESP32 RSA) or a shift register (TI MSP). A few chips with undocumented cryptographic accelerators were included as well (SAML11, RA2E1) of which we will reverse engineer the functioning, but doing this for every chip with undocumented accelerator is impractical.

The NXP i.MX series of microprocessors feature a 'CAAM' cryptographic accelerator, though its documentation is only available under NDA. However, besides CAAM, these chips also contain an HDMI encoder with hardware to perform HDCP (HDMI video DRM) cryptography. The HDCP key registers are readable, hence a PKO attack may not be necessary here.

The NXP LPC55S6x series also contains multiple accelerators: CASPER for RSA and ECC, PRINCE for flash memory contents, and an AES accelerator. CASPER, the public-key accelerator, only operates on units of 64 bits. It thus has to be called many times to perform a single full public-key operation. Hence, it cannot store a full private key. This limited design also makes it likely that its initial inputs are overwritten by temporary values. PRINCE only uses keys sourced from the system PUF. The AES accelerator uses the same MMIO registers for key and data input, rekeying is only possible by completely resetting the AES peripheral. With neither of these accelerators, PKO attacks seem possible.

Most other chips not further treated in this study (such as the NXP i.MX HDMI encoder, the TI MSP432E as well as several STM32G4 and STM32Fx chips from STMicroelectronics) have readable key registers. While they do not support TrustZone-M, they still have ‘privilege separation’ mechanisms, ranging from proprietary IPE features (‘IP protection’ and ‘PCROP’, respectively) to, in the case of the STM32, a debugger flash readout protection feature (RDP level 1). In these cases an attacker operating under the model defined in Section 3.1 may be able to directly read the key (otherwise stored in protected non-volatile storage) from the cryptographic accelerator, hence a PKO attack would not be relevant in these chips.

Manufacturer	**Name**	**Ciphers**	**Priv. sep.**	**Used**	**Notes**
Espressif	ESP32S2, S3, C3	RSA	Encrypted keys	✔	AES key registers readable, thus focus on RSA
Microchip	SAML11	AES	TZ-M	✔	only access through ROM API
Renesas	RA2E1	AES	‘Security MPU’	✔	No docs, reverse engineer
	RA4M2	AES	TZ-M	✔	No docs, reverse engineer
ST	STM32G0, STM32Fx	AES	PCROP	✔	Write-only key regs
	STM32L5	AES, RSA, ECC	TZ-M	✔	Write-only key regs
	STM32U0	AES	HDP	✔	Write-only key regs
	STM32H5, U5	AES, RSA, ECC	TZ-M	✔	Has mitigations for RSA & ECC, cf. [27]
	STM32MP1	AES	TZ-A	✔	Write-only key regs
TI	MSP430FR59	AES	IPE	✔	Key store is shift register
	MSP432P	AES	IPE	✔	same as MSP430
Microchip	ATXmega AU	AES, DES	N/A	✘	Key registers readable
NXP	i.MX 8M	AES, RSA, ECC	TZ-A	✘	Key registers readable (HDMI encoder, cf. [18])
	LPC55S6x	AES, RSA, ECC	TZ-M	✘	Countermeasures (cf. [19])
ST	STM32G4, STM32Fx	AES	PCROP	✘	Key registers readable (cf. e.g. [26])
	STM32H7	AES	PCROP	✘	Known vulnerability [12]

TI	MSP432E	AES	IPE	✘	Key registers readable (cf. [31])
GigaDevice	GD32L23	AES	N/A	✘	No priv. sep. to attack
	GD32VW55x	AES	key from OTP	✘	Priv. sep. trivially not vulnerable
Infineon	PSoC 64	AES	N/A	✘	No priv. sep. to attack (only accessible from high-privilege M0+)
Microchip	SAML2x	AES	N/A	✘	No priv. sep. to attack (but vulnerable key registers)
TI	SimpleLink, MSPM0	AES	N/A	✘	No priv. sep. to attack
	MSPM0Lx22x	AES	N/A	✘	Countermeasures (cf. [32]), chip not available
Allwinner	sun4i, 8i	AES	N/A	✘	No docs
Microchip	PIC32CMLx	AES	TZ-M	✘	No docs, only accessible through SDK
NXP	i.MX 6/7/8	AES, RSA, ECC	TZ-A	✘	No docs (‘CAAM’ module), NDA required
	i.MXRT10xx	AES	secure boot	✘	No docs (‘DCP’ module), NDA required
Renesas	RA, RX	AES	TZ-M (some)	✘	No docs, NDA required
Rockchip	RK3399, RK3588	AES	TZ-A	✘	No docs
TI	AM Sitara	AES	TZ-A	✘	No docs, NDA required
Xilinx	ZynqMP	AES	TZ-A	✘	No docs, NDA required

Table 2: List of all chips considered in this survey. The column ‘ciphers’ denotes the ciphers supported by the accelerators which were deemed relevant to this work. The column ‘priv. sep.’ lists the name of the privilege separation mechanism present in the microcontroller. ‘TZ-A’ and ‘TZ-M’ mean TrustZone-A and -M, respectively. ‘PCROP’, ‘HDP’ and ‘IPE’ are proprietary mechanisms that disallow read access to certain regions of code flash, c.f. [5, 23]. The column ‘used’ shows whether this chip was then further investigated in this survey for the presence of vulnerabilities against PKO attacks.

4.4 Results and Analysis

Table 3 presents a summary of the results of the survey.

Manufacturer	**Name**	**Year**	**Vuln.**	**Cplx.**	**Notes**
TI	MSP430FR5994	2016	✔	$16 \cdot 2^8$	Cf. Section 4.4.5
	MSP432P401R	2015	✔	$16 \cdot 2^8$	same accel. as MSP430
ST	STM32L562 (AES)	2018	✔	$4 \cdot 2^{32}$	Cf. Section 4.4.1
	STM32G061	2018	✔	$4 \cdot 2^{32}$	Same accel. as L5
	STM32U083	2024	✔	$4 \cdot 2^{32}$	Same accel. as L5
	STM32MP157	2019	✔	$4 \cdot 2^{32}$	Same accel. as L5
	STM32H730	2019	✔	$4 \cdot 2^{32}$	Known vuln. [12]
	STM32L562 (RSA/ECC)	2018	?	N/A	Cf. Section 4.4.1
	STM32U5A5, H5 (RSA/ECC)	2021-2023	?	N/A	Cf. Section 4.4.1
Espressif	ESP32S2, S3, C3	2020-2021	✔/✘	P-H	Cf. Section 4.4.2
Microchip	SAML21	2015	N/A	N/A	Technically out of scope, cf. Section 4.4.3
	SAML11	2018	✘	N/A	Cf. Section 4.4.3
Renesas	RA2E1	2020	✘	N/A	Cf. Section 4.4.4
	RA4M2	2019	✘	N/A	Cf. Section 4.4.4
ST	STM32U5A5, H5 (AES)	2021-2023	✘	N/A	Cf. Section 4.4.1

Table 3: List of all evaluated chips, and an indication of whether their accelerators are vulernable to PKO attacks or not (and if so, notes the complexity of the offline computations required for the attack, ‘P-H’ denotes use of the Pohlig-Hellman algorithm).

Some of the results for particular chips are unclear or require further elaboration. These notes are listed below:

4.4.1. ST STM32 series The STM32L5, G0, MP1 and U0 all share the same AES accelerator, hence these are all equally vulnerable. Separate key words can be updated independently as long as the `EN` bit in the control register is set to zero.

The accelerator shared between the STM32U5 and H5 is based on the former one, but went through an Arm Platform Security Architecture (PSA) level 3 certification process [27]. Countermeasures against PKO attacks have now been implemented as well. This is reflected in the register interface, with an extra `KEYVALID` bit in the status register.

However, the public-key accelerator ('PKA') of the STM32 series is more troublesome. People have reported[5] having issues with getting the accelerator to work at all in the first place. We also faced this problem in our experimentation, and are thus unable to determine whether it is vulnerable to PKO attacks.

The STM32U5 extends the L5 PKA by adding different commands for computations using either the public or private key (automatically erasing remanent private key data in the latter case), while the L5 does not distinguish between the two. Hence, if the correct operation is used, the STM32U5 PKA should not be vulnerable to PKO attacks.

4.4.2. ESP32 The ESP32 RSA accelerator can be used in two modes: the 'normal' mode in which the accelerator is accessed directly and keys are supplied by the CPU, and the 'Digital Signature' (DS) mode in which the CPU uploads an encrypted key blob to the accelerator. The accelerator autonomously decrypts this key blob using a key from the one-time-programmable (OTP) memory, this key is not visible to the CPU.

In the 'normal' mode, the MMIO registers that store the RSA parameters are write-only. However, every parameter on its own (e.g. the modulus) can be changed without necessarily invalidating the others. Hence, a Pohlig-Hellman-based attack (c.f. Section 3.3) is possible. Though, the practicality of such an attack is questionable, as there is no memory protection measure (like PCROP or IPE) able to protect a key from an attacker-controlled CPU. Furthermore, the default Espressif driver code for the RSA peripheral immediately powers down said peripheral, erasing the keys stored in MMIO registers.

In the DS mode, there again seems to be an issue with the main functioning of the accelerator, and we struggled with inconsistent results here as well[6]. However, the output of the accelerator was still *deterministic*, unlike in the STM32L5. Although, in this case, overwriting only the base and modulus of the accelerator after a DS operation always results in an output of zero. Hence, it seems that after a DS operation, the MMIO registers for the RSA parameters are always cleared. Thus, in DS mode, the ESP32 RSA accelerator is not vulnerable to PKO attacks.

[5] `https://community.st.com/t5/stm32-mcus-security/pka-modular-exponentiation-not-working-as-expected/m-p/699510` [6] As reported at e.g. `https://esp32.com/viewtopic.php?f=12&t=42057`

4.4.3. Microchip SAML11 The SAML11 has an AES accelerator, though the only documented way to make use of it is to use a runtime utility function which resides in the boot ROM. The contents of the boot ROM cannot be read by regular user code.

Using timer interrupts to trace execution flow (using the same method as described in [5, Sec. 3.3]), it is possible to obtain some information on the structure of the boot ROM code. We considered using DMA to create extra bus contention as a timing side channel leaking information on which memory is accessed at which point (cf. [6]). However, this is not useful, as the cryptographic accelerator hardware is attached to the Cortex-M23's single-cycle I/O bus, which is exclusively controlled by the CPU.

The entire AES key schedule appears to be calculated entirely in software, with the cryptographic acceleration MMIO peripheral seemingly only responsible for the `SubBytes`, `ShiftRows` and `MixColumns` steps of AES, one round at a time. Hence, the key never leaves the CPU registers and does not get stored in any MMIO register. The SAML11 ROM AES routines are thus not vulnerable to a PKO attack.

Note that the older SAML2x series has an AES accelerator of which the key register allows partial overwrites. However, this older series does not contain any privilege separation mechanism (such as TrustZone, 'IP encapsulation', or a debug lockout level that disallows flash access but allows MMIO access) for an attack to be meaningful.

4.4.4. Renesas RA2E1 and RA4M1 Cryptographic accelerator designs in Renesas microcontrollers are often shared between multiple chips. These accelerators exist in several 'versions' or 'generations'. RA2xx MCUs have simple AES and TRNG peripherals, while RA4xx and up have a more comprehensive 'Secure Crypto Engine' (SCE). Different 'generations' of SCE exist, numbered 5, 5B, 7 and 9. The SCE peripherals support more ciphers (sometimes asymmetric ones as well) and block cipher modes of operation, as well as key 'wrapping'. Renesas does not publish documentation on the cryptographic accelerators in its microcontrollers. However, they do publish *obfuscated* driver code as part of their 'Flexible Software Package' (FSP).

For the RA2xx series, reverse-engineering this code revealed a rather standard AES accelerator, though it does incorporate PKO countermeasures (as evidenced by bits 24 and 25 in the 32-bit register at offset +4). However, there seems to be a 'secret' bit (#15) in this register that is unused by the FSP driver code. Setting this bit seems to cause a time-sensitive perturbation, where sometimes the result of a cryptographic operation is not propagated to the output registers. The exact functioning of this bit and its security implications are unclear.

The SCE peripheral (in RA4xx and up) has a 'command interface' where a 16-*byte* magic number must be supplied to perform a command, such as loading a key, setting the mode of operation, etc. The peripheral also mandates the use of 'wrapped' keys. 'Wrapped' keys are encrypted using an on-chip key residing inside the SCE hardware [10]. Next to this encryption layer, they are protected by a MAC as well, to protect against possible tampering [11]. A 'wrapped' key is thus twice the size of a plain key: an 128-bit AES key becomes 256 bits in size when wrapped, an AES-256 plain key turns into a 512-bit wrapped key.

Any key passed to the SCE meant for cryptographic operations must be supplied in its 'wrapped' form. These cannot be tampered with, hence PKO attacks are not possible on RA4xx/6xx/8xx MCUs either.

4.4.5. TI MSP430, MSP432 and MSPM0 This subsection concerns itself with the separate MSP430, MSP432 and MSPM0 lines of microcontrollers, all from Texas Instruments. While they have different CPU cores and designs, they do share some peripherals, including the AES accelerator. TI MSP chips use a key MMIO register that acts as a shift register, as opposed to making any key word separately addressable as done on e.g. the STM32 series.

The AES accelerator keeps track of whether the key is fully written through the `AESKEYWR` bit in the `AESASTAT` register. However, this bit is writeable, and hence partial keys will be accepted as well.

Many MSP430 chips as well as the MSP432P incorporate some form of an 'IP protection' feature, as detailed in [5]. Most MSPM0 chips do *not* support this, except for the newer MSPM0Lx22x and MSPM0G351x models — currently the only PSA-certified microcontrollers in the MSPM0 series. However, these models instead incorporate an 'ADVAES' accelerator, which, according to their reference manual, does protect against PKO attacks [32]. At the time of writing, these MCUs are not yet available, and thus statements from the reference manual could not be verified.

5 Countermeasures

It should hopefully be clear by now that without countermeasures against PKO attacks, having write-only instead of read-write key registers makes little difference. However, care must be taken when implementing hardware countermeasures. The internal state machine that detects partially-overwritten keys must function regardless of the MMIO register access pattern, and there should be no 'force OK' bit (such as in the MSP430), yet it is possible to do this correctly (as seen in the STM32U5 and Renesas RA2E1). When designing the register interface of a cryptographic accelerator, a command-based interface (such as seen in the Renesas SCE) with which only a full key can be submitted at once may be a more prudent design option.

When using a vulnerable accelerator in a microcontroller, firmware engineers can mitigate PKO attacks in several ways. This mainly relies on redrawing the border between secure and nonsecure system components such that the accelerator falls entirely within the area marked as secure, instead of straddling the border. For example, if a TEE such as TrustZone-M is available, the cryptographic accelerator should be configured to only allow accesses from within TZ secure code (and expose a cryptographic software API to nonsecure code if needed). If no TEE exists in the chip, disabling interrupts during the operation of the accelerator and clearing the key registers afterwards should suffice against attackers taking over the CPU, though this could incur extra CPU time overhead. Chips with different debug readout protection levels should be configured to use the highest protection level (though downgrade attacks exist for this, e.g. [20] in the STM32 series).

6 Conclusion

As shown in Section 4.4, several off-the-shelf microcontrollers available on the market from different vendors are vulnerable to PKO attacks. Despite a lack of attention from academia, real-world attackers have demonstrated a sustained interest in these vulnerabilities. Note, however, that some chips included in the survey are in a worse situation where an adversary does not even need to use a PKO attack to extract keys from cryptographic accelerators, while those keys would still otherwise reside in protected nonvolatile memory.

Luckily, microcontroller manufacturers seem to move towards incorporating countermeasures: after having brought vulnerable chips to market, Microchip, TI and ST have all released newer models with countermeasures against PKO attacks. This seems to be correlated with the creation of the Arm PSA certification standard, of which the specifications were released in 2017. This could be an indication that such certification efforts increase the security of consumer products, though it is unclear to what extent PSA certification has actually influenced these matters: the only publicly available documentation with a certain degree of details belongs to level 1 certification only, and it is unclear which exact requirements are maintained by certification laboratories.

Code used for testing microcontrollers and microprocessors for vulnerability against PKO attacks can be found at `https://anonymous.4open.science/r/pko-testcode-5D2D`.

Disclosure of Interests The authors have no competing interests to declare that are relevant to the content of this article.

References

1. 3dbrew, 3DS System Flaws, Archived at `https://web.archive.org/web/20240829223013/https://www.3dbrew.org/wiki/3DS_System_Flaws#Hardware`. `https://www.3dbrew.org/wiki/3DS_System_Flaws%5C#Hardware` (visited on 09/08/2024)
2. Albrecht, M.R., Haller, M., Mareková, L., Paterson, K.G.: Caveat Implementor! Key Recovery Attacks on MEGA, Cryptology ePrint Archive, Paper 2023/329 (2023). `https://eprint.iacr.org/2023/329`.
3. Backendal, M., Haller, M., Paterson, K.G.: MEGA: Malleable Encryption Goes Awry, Cryptology ePrint Archive, Paper 2022/959 (2022). `https://eprint.iacr.org/2022/959`.

4. Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: Kaliski, B.S. (ed.) Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology. CRYPTO '97, pp. 513–525. Springer-Verlag, Berlin, Heidelberg (1997). https://doc.lagout.org/security/Papers/DFA%20of%20Secret%20Key%20Cryptosystems.pdf
5. Bognár, M., Magnus, C., Piessens, F., Van Bulck, J.: Intellectual Property Exposure: Subverting and Securing Intellectual Property Encapsulation in Texas Instruments Microcontrollers. In: 33rd USENIX Security Symposium (USENIX Security 24), pp. 2155–2172. USENIX Association, Philadelphia, PA (2024). https://www.usenix.org/conference/usenixsecurity24/presentation/bognar
6. Bognár, M., Van Bulck, J., Piessens, F.: Mind the Gap: Studying the Insecurity of Provably Secure Embedded Trusted Execution Architectures. In: 2022 IEEE Symposium on Security and Privacy (SP), pp. 1638–1655 (2022). https://doi.org/10.1109/SP46214.2022.9833735. https://mici.hu/papers/bognar2022gap.pdf
7. Bruseghini, L., Huigens, D., Paterson, K.G.: Victory by KO: Attacking OpenPGP Using Key Overwriting. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. CCS '22, pp. 411–423. Association for Computing Machinery, Los Angeles, CA, USA (2022). https://doi.org/10.1145/3548606.3559363
8. Clavier, C.: Secret External Encodings Do Not Prevent Transient Fault Analysis. In: Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems. CHES '07, pp. 181–194. Springer-Verlag, Vienna, Austria (2007). https://doi.org/10.1007/978-3-540-74735-2_13. https://iacr.org/archive/ches2007/47270181/47270181.pdf
9. Domke, F.t.: Almost Secure, Archived at https://web.archive.org/web/20241119012300/https://debugmo.de/2011/11/almost-secure/. (2011). https://debugmo.de/2011/11/almost-secure/ (visited on 11/21/2024)
10. Electronics, R.: RA6M5 Group: User's Manual: Hardware, (2023). https://www.renesas.com/en/document/man/ra6m5-group-users-manual-hardware?r=1493931 (visited on 12/17/2024)
11. Electronics, R.: Renesas RA Family Application note: Renesas Security Engine Operational Modes, (2024). https://www.renesas.com/en/document/apn/renesas-security-engine-operational-modes?r=1493931 (visited on 12/17/2024)
12. GaryoderNichts, Looking into the Nintendo Alarmo, Archived at https://archive.ph/4nhB8. (2024). https://garyodernichts.blogspot.com/2024/10/looking-into-nintendo-alarmo.html (visited on 10/31/2024)
13. Kerins, T., Kursawe, K.: A Cautionary Note on Weak Implementations of Block Ciphers, (2006). PDF available at https://web.archive.org/web/20221206234645/https://www.cosic.esat.kuleuven.be/wissec2006/papers/10.pdf.
14. Klima, V., Rosa, T.: Attack on Private Signature Keys of the OpenPGP Format, PGP(TM) Programs and Other Applications Compatible with OpenPGP, Cryptology ePrint Archive, Paper 2002/076 (2002). https://eprint.iacr.org/2002/076.
15. Korth, M.: GBATEK DSi AES I/O Ports, Archived at https://web.archive.org/web/20220829222918/http://problemkaputt.de/gbatek-dsi-aes-i-o-ports.htm. (2021). https://problemkaputt.de/gbatek-dsi-aes-i-o-ports.htm (visited on 10/04/2022)
16. Lu, Y.: The 3DS Cryptosystem, Archived at https://web.archive.org/web/20240625001543/https://yifan.lu/2016/04/06/the-3ds-cryptosystem/.

(2016). https://yifan.lu/2016/04/06/the-3ds-cryptosystem/ (visited on 09/08/2024)

17. Minerva: The curse of ECDSA nonces : Systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(4), 281–308 (2020). https://doi.org/10.13154/tches.v2020.i4.281-308. https://tches.iacr.org/index.php/TCHES/article/view/8684
18. NXP, i.MX 8M Dual/8M QuadLite/8M Quad Applications Processors Reference Manual, Sign-in required for download. (2021). https://www.nxp.com/webapp/Download?colCode=IMX8MDQLQRM
19. NXP, LPC55S6x/LPC55S2x/LPC552x User manual, Sign-in required for download. (2021). https://www.nxp.com/webapp/Download?colCode=UM11126
20. Obermaier, J., Tatschner, S.: Shedding too much Light on a Microcontroller's Firmware Protection. In: 11th USENIX Workshop on Offensive Technologies (WOOT 17). USENIX Association, Vancouver, BC (2017). https://www.usenix.org/conference/woot17/workshop-program/presentation/obermaier
21. Pohlig, S., Hellman, M.: An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (Corresp.) IEEE Transactions on Information Theory **24**(1), 106–110 (1978). https://doi.org/10.1109/TIT.1978.1055817. https://www-ee.stanford.edu/~hellman/publications/28.pdf
22. Rodrigues, C., Oliveira, D., Pinto, S.: BUSted!!! Microarchitectural Side-Channel Attacks on the MCU Bus Interconnect. In: 2024 IEEE Symposium on Security and Privacy (SP) (2024). https://bustedattack.com/resources/BUSted.pdf
23. Schink, M., Obermaier, J.: Taking a look into execute-only memory. In: Proceedings of the 13th USENIX Conference on Offensive Technologies. WOOT'19, p. 1. USENIX Association, Santa Clara, CA, USA (2019). https://www.usenix.org/system/files/woot19-paper_schink.pdf
24. SciresM, hexkyz, Je Ne Sais Quoi - Falcons over the Horizon, Archived at https://archive.is/wNT42. (2021). https://hexkyz.blogspot.com/2021/11/je-ne-sais-quoi-falcons-over-horizon.html (visited on 12/06/2024)
25. plutoo derrek smea , s.: Console Hacking: Breaking the 3DS, (2015). https://media.ccc.de/v/32c3-7240-console_hacking (visited on 12/16/2024)
26. STMicroelectronics, RM0440 Reference manual: STM32G4 series advanced Arm-based 32-bit MCUs, (2024). https://www.st.com/resource/en/reference_manual/rm0440-stm32g4-series-advanced-armbased-32bit-mcus-stmicroelectronics.pdf (visited on 12/08/2024)
27. STMicroelectronics, RM0456 Reference manual: STM32U5 Series Arm-based 32-bit MCUs, (2023). https://www.st.com/resource/en/reference_manual/rm0456-stm32u5-series-armbased-32bit-mcus-stmicroelectronics.pdf (visited on 11/18/2024)
28. Swierczynski, P., Fyrbiak, M., Koppe, P., Paar, C.: FPGA Trojans Through Detecting and Weakening of Cryptographic Primitives. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **34**(8), 1236–1249 (2015). https://doi.org/10.1109/TCAD.2015.2399455
29. SwitchBrew, Switch System Flaws, Archived at https://web.archive.org/web/20240826033554/https://switchbrew.org/wiki/Switch_System_Flaws#Hardware. https://switchbrew.org/wiki/Switch_System_Flaws%5C#Hardware (visited on 09/08/2024)
30. Tan, S.J., Bratus, S., Goodspeed, T.: Interrupt-Oriented Bugdoor Programming: A Minimalist Approach to Bugdooring Embedded Systems Firmware. In: Proceed-

ings of the 30th Annual Computer Security Applications Conference. ACSAC '14, pp. 116–125. Association for Computing Machinery, New Orleans, Louisiana, USA (2014). https://doi.org/10.1145/2664243.2664268

31. Texas Instruments, MSP432E4 SimpleLink Microcontrollers Technical Reference Manual, (2018). https://www.ti.com/lit/ug/slau723a/slau723a.pdf (visited on 12/08/2024)
32. Texas Instruments, MSPM0 L-Series 32MHz Microcontrollers Technical Reference Manual, (2024). https://www.ti.com/lit/pdf/slau847 (visited on 11/18/2024)
33. Yen, S.-M., Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. IEEE Transactions on Computers **49**(9), 967–970 (2000). https://doi.org/10.1109/12.869328. https://marcjoye.github.io/papers/YJ00chkb.pdf
34. Yuce, B., Schaumont, P., Witteman, M.F.: Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation. Journal of Hardware and Systems Security **2**, 111–130 (2018). https://arxiv.org/pdf/2003.10513