IACR Transactions
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–26. DOI:XXXXXXXX

Security and Trust in Open Source Security Tokens

Marc Schink, Alexander Wagner, Florian Unterstein and Johann Heyszl

Fraunhofer Institute for Applied and Integrated Security (AISEC), Germany,
firstname.surname@aisec.fraunhofer.de

Abstract. Using passwords for authentication has been proven vulnerable in countless security incidents. Hardware security tokens effectively prevent most password-related security issues and improve security indisputably. However, we would like to highlight that there are new threats from attackers with physical access which need to be discussed. Supply chain adversaries may manipulate devices on a large scale and install backdoors before they even reach end users. In evil maid scenarios, specific devices may even be attacked while already in use. Hence, we thoroughly investigate the security and trustworthiness of seven commercially available open source security tokens, including devices from the two market leaders: SoloKeys and Nitrokey. Unfortunately, we identify and practically verify significant vulnerabilities in all seven examined tokens. Some of them are based on severe, previously undiscovered, vulnerabilities of two major microcontrollers which are used at a large scale in various products. Our findings clearly emphasize the significant threat from supply chain and evil maid scenarios since the attacks are practical and only require moderate attacker efforts. Fortunately, we are able to describe software-based countermeasures as effective improvements to retrofit the examined devices. To improve the security and trustworthiness of future security tokens, we also derive important general design recommendations.

Keywords: security token · second factor authentication · FIDO · fault injection attack · side-channel attack · firmware protection

1 Introduction

Passwords are the most common authentication mechanism for private as well as professional IT services such as social media, banking or enterprise infrastructure. Unfortunately, the use of passwords leads to many security issues. As many as 81 % of all hacking-related data breaches are the result of stolen or weak passwords [Inc17]. Weak passwords allow for guessing or dictionary attacks, password re-use may lead to *credential stuffing* where stolen passwords are used for other services of the same users. Passwords may get stolen using phishing attacks [Pre20] or through keylogger malware [SG17]. Even large enterprises with sufficient resources are prone to such threats [Bis19, Ike20, Sue12, Gib16]. Password managers aim to improve password strength and password diversification by relieving the user of the burden to memorize many complicated passwords. There are also ways to improve password security on the service side by restricting the number of login attempts and user alerts on logins from unknown devices. In all cases, however, passwords are easily transferable, once recovered, and can be used by attackers from anywhere on earth.

Hardware security tokens as second authentication factor or *passwordless* logins [Cor18] have been designed to prevent the described threats effectively. They are small dedicated devices which provide a means for authentication using strong cryptography and a securely stored key. Only users in physical possession of such a token, often referred to as *something*

you have, may authenticate. Typically, such devices only consist of a microcontroller or security controller and a small amount of firmware. Hence, the required trusted code base and hardware is reduced to this relatively simple device, and any software on the computer or smartphone is eliminated from the trusted code base. Even if such a token is permanently plugged into a user's computer, the above mentioned attacks on password-based authentication are successfully mitigated. Security tokens are already used broadly in research institutions, big tech companies and governmental agencies [Nit]. As one of the largest tech companies, Google internally requires the use of security tokens [Kre18]. Building on this, Google is currently marketing its own closed source tokens and pushing towards open source alternatives for the future with the help of its OpenSK project [Ope73].

But while severe issues with password security are effectively taken care of, which is a big step forward, there are new issues that need to be discussed. As a severe threat we would like to highlight that the whole value chain including the designs, internationally distributed productions, the supply chain and the final distribution to customers of tokens is a new attack surface for malicious manipulation. The National Security Agency (NSA) had for example intercepted and compromised network equipment on the distribution from the manufacturer to end users [Cox14]. For security tokens, such manipulation are even simpler. An attacker may for instance act as a retailer on a platform like Amazon and sell compromised tokens at a large scale. Users will have little to no means to recognize such manipulations. Another threat for high-profile individuals stems from so-called evil maid scenarios, where an attacker gains short-time physical access to a token.

In summary, the trustworthiness of security tokens for users is a major issue which is why we investigate them in this contribution. We specifically concentrate on security tokens which are marketed as open source. Open source designs generally facilitate trustworthiness because they allow security evaluations through a broad community of independent security researchers. We think that open source security tokens will more likely achieve higher trustworthiness in the longer term. Since there are no open source chips available, even open source tokens are based on chips with closed source designs such as commercial off-the-shelf (COTS) microcontrollers or certified security controllers where parts of the software, such as the operating system, are closed source.

Previous security research has focused on individual closed source tokens [Roc21, RP13, Yub19, Yub17] or on open source tokens with a focus on firmware only [Sol20, CS20, O'F19]. To date, there is no comprehensive analysis of the security of open source tokens with respect to physical attacks which are highly relevant with supply chain adversaries. These attacks get even more relevant with the FIDO2 standard which introduces the so-called *passwordless login* [Cor18]. Thus, merely compromising the token, for example, through a backdoor is sufficient for an attacker to authenticate against any service.

We focus on hardware attacks in supply chain and evil maid scenarios and perform a systematic analysis of the most important commercially available candidates, including tokens from SoloKeys [Sol] and Nitrokey [Nit]. We also include the OpenSK research project from Google because it is used by the FEITIAN OpenSK Dongle [FEI21] and it may also be relevant for future open source security tokens. Our paper provides the following contributions:

- Systematic security analysis of seven commercially available open source security tokens under a realistic adversarial model from supply chain and evil maid scenarios.
- Discovery and practical demonstration of novel non-invasive attacks against all devices that allow to clone a token or inject malicious firmware, which include:
  - A non-invasive EM fault injection attack to bypass the read-out protection of the STM32L4 and the nRF52840 microcontroller.
  - An EM side-channel attack against the widely deployed cryptography library micro-ecc.

  - A timing side-channel attack against the ARM CryptoCell 310, the on-chip accelerator for ECC used by the nRF52840 microcontroller.
  - An attack on the inter-chip communication on the Nitrokey FIDO U2F.
- A systematic approach to use side-channel leakage to narrow down the parameter space for fault injection attacks by several orders of magnitude.
- Software-based mitigations and countermeasures for all identified vulnerabilities that can be retrofit to ensure the tokens security despite physical attacks.
- Design recommendations and guidelines derived from our findings to increase the security and trustworthiness in future open source security tokens.

2 Trust Issues with Security Tokens

Unlike choosing a password where users have full control over its strength, a security token is a mere black box to users. As an inevitable downside to the general security improvement, users need to trust the entire value chain of such tokens. What does this mean and how do we investigate this?

1. Users need to trust that the token manufacturer designs and implements a secure device software without any unintentional vulnerabilities or backdoors, such as weak random number generators. This needs to be investigated which is difficult for closed source designs. In case of an open source design or open source design parts, this trust can be improved through public review from independent third parties.
2. Users need to trust that the manufacturer selects hardware components such as microcontrollers which provide the required security features and which do not contain any unknown vulnerabilities. We investigate this for all selected tokens through testing relevant attacks during our security analyses.
3. Users need to trust that the token is assembled correctly, contains the trusted software and that the token is not manipulated during the supply and distribution chain. Additionally, users must either keep a token within sight at all times or trust that no one performs any manipulations in an evil maid scenario. We take this into account by using a respective adversarial model for the security analyses.

In order to reduce the necessary trust of users in FIDO-based security tokens, Dauterman et al. [DCM+19] proposed True2F. This approach is a modification of the Universal 2nd Factor (U2F) protocol such that a user's browser also contributes to the entropy of generated authentication keys. This ensures that generated keys have a certain minimum entropy even in case of a manipulated token. Even though True2F aims at preventing backdoors in security tokens, it only covers manipulations targeting the key generation process. In particular, it does not prevent or detect malicious security tokens. For example, a backdoor in the firmware that allows an adversary to extract the credentials in an evil maid scenario is still feasible.

A more comprehensive approach to establish trust in security tokens is to open source their entire firmware and design. This generally facilitate investigations and allows evaluations through independent security researchers. Some work has already been done to identify vulnerabilities and subsequently improve the respective security designs. For instance, the architecture and implementation of the Solo from SoloKeys has been notably reviewed by independent third parties. O'Flynn [O'F19] published an EMFI attack on the USB stack in 2019, which allows to read out secret key material. In 2020, SoloKeys itself initiated a security review [Sol20] which was later published. A vulnerability was described

which allows to update the firmware with an older and outdated signed firmware image bypassing the downgrade protection. Collin et al. [CS20] reviewed the side-channel security of the HMAC-SHA256 implementation of the Solo's firmware but found no vulnerabilities. We want to further resolve trust issues with open source security tokens by performing a broad investigation of currently available products.

3 Adversarial Model and Methodology

The final goal of an attacker is to gain unrestricted access to the services for which the token is used as means of authentication. This is achieved if the respective cryptographic keys or passwords can be extracted.

3.1 Adversarial Model

We define an adversarial model which is in line with the previously identified trust issues and that we deem reasonable for this application. This means that we concentrate on supply chain and evil maid adversaries which try to exploit design vulnerabilities. The security tokens we investigate did not undergo any formal product certification including respective adversarial model definitions. Neither their resistance to physical attacks nor any other kind of attacks is documented.

Know-How We assume an attacker with considerable technical know-how and software tooling. The attacker may use single devices to gain knowledge through more extensive reverse engineering and profiling efforts.

Physical Access and Time Consistent with supply chain or evil maid scenarios we allow the attacker time-limited physical access to tokens for the actual attacks, for example in the range of one or several hours. Physical proximity is inevitable because security tokens implement means, such as a button press, to verify user presence. However, attacks must not leave any visible marks which would alert a user of the security breach. A longer time or even permanent theft of a token could be detected by the user and the device would be revoked from all registered services rendering an attack pointless. Granted that temporal possession as evil maid already allows short-period use of a token, our investigation focus is on attacks that may lead to secret extraction for unlimited use.

Equipment and Effort The nature of either an evil maid scenario or a high-volume supply chain attack scenario means that only lower-end equipment and limited attacker efforts are reasonable. We consider attack equipment that does not exceed a couple thousand USD in costs and can likely be operated in a portable manner, such as logic analyzers, compact side-channel measurement or fault injection setups. Equipment for high-precision measurements is usually not portable and requires extensive device preparation efforts. Hence, high-end attacks are unrealistic and need not be considered. Therefore, it is in fact a fair design choice to include COTS microcontrollers which are neither marketed nor designed as dedicated high security products withstanding physical attacks. Nonetheless, security tokens based on such microcontrollers still need to withstand attacks according to this adversarial model.

3.2 Methodology

We evaluate all security tokens according to the same top-down methodology which means that attacks with the most favorable properties for adversaries are discussed and tested first. In each evaluation step, the demands on the attacker in terms of required device-specific

Table 1: Analyzed security tokens including their hardware and firmware versions.

Manufacturer	Device	Hardware	Firmware
SoloKeys	Solo	v2.1	4.0.0
	Somu	v1.0	
Nitrokey	Nitrokey FIDO U2F	Rev. 9	nk-v1.1
	Nitrokey Pro 2	Rev. 3	v0.10
	Nitrokey FIDO2	Rev. 5	2.0.0.nitrokey
Purism	Librem Key	Rev. 3	v0.10
Google	OpenSK	-	-
FEITIAN	OpenSK Dongle	v1 / v2	

knowledge, attack effort and equipment costs increase. All considered attacks must fit the adversarial model which means that the methodology stops well before high-end attacks.

Architecture and Interfaces The investigation starts with an evaluation of the device architecture and by identifying components which are used for the storage of secrets and actual (cryptographic) authentication operations. If a token architecture consists of multiple chips, their exchange of data is evaluated. Low-cost equipment such as logic analyzers are used to reverse engineer the communication protocols, if necessary, and to intercept the communication for possible inter-chip attacks. As a next step, debug interfaces and other hardware interfaces of chips are evaluated for ways of compromising the secrets. Development tools for microcontrollers such as debug probes are employed. Note that we do not specifically investigate software communication stacks for the USB interface, for example in regard to memory boundary issues, because they have already been analyzed thoroughly [O'F19, Sol20] and tested automatically [Ope73].

Cryptographic Operations If previous attempts fail, cryptographic implementations that operate on or generate secret keys are investigated for attacks within the adversarial model. Pen-like EM measurement probes with coarse spatial resolution and oscilloscopes operating in a sub-GHz range are reasonable equipment here. No device preparation such as decapsulation is considered and allowed measurement times during an attack are relatively short (profiling may be more extensive). This limits the range of attacks to simpler ones such as timing side-channels attacks and simple power analysis (SPA) attacks exploiting non-uniform executions for example. In terms of fault injection equipment, only non-invasive fault injection is reasonable and the number of attempts is time-limited.

Hardware Security Features After an analysis of the interfaces and of the cryptographic operations, the hardware security features of the included chips are examined in more detail. Every vulnerability that may be uncovered through more extensive one-time analysis of a respective chip may possibly lead to effective attacks in the field. In this step, we evaluate if included chips exhibit exploitable vulnerabilities in an experimental fashion. Some reverse engineering effort is often needed due to the black box nature of the implemented security measures. At this stage, side-channel and fault injection equipment may also be used.

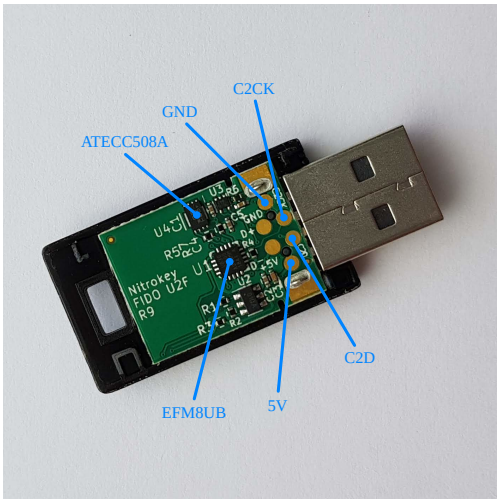


Figure 1: Nitrokey FIDO U2F with open housing. The integrated chips and accessible supply and debug pins are highlighted.

4 Selection of Security Tokens

We include all relevant open source security tokens that have commercially been available at the time of writing. This includes devices from the two market leaders Nitrokey and SoloKeys which are used by numerous well-known technology companies and governmental agencies [Nit]. Further, we include the OpenSK which is a research platform initiated by Google and already used in products like the OpenSK Dongle manufactured by FEITIAN. The comprehensive device examination serves to understand the individual as well as broader shortcomings and vulnerabilities that need to be improved in state-of-the-art security tokens. All examined devices together with their respective hardware and firmware revisions analyzed are listed in Table 1.

5 Nitrokey FIDO U2F

The first product we examine is the Nitrokey FIDO U2F which is a dedicated hardware security token that implements the U2F specification. This product is based on the no longer maintained U2F Zero project [Zerc4]. Figure 1 shows the token with an open housing. The token consists of an EFM8UB microcontroller from Silicon Labs and an ATECC508A secure element from Microchip. Both chips are connected via an I2C bus. The microcontroller provides the U2F API to the host via USB. Since the secure element cannot handle the entire U2F functionality, the microcontroller steers the operation flow and must also process sensitive data. Hence, in addition to the secure element, the microcontroller and the I2C bus are sensitive to attacks as well.

In line with the first step in our investigation methodology, we analyzed the device on the architecture and interface level. A simple exchange of the hardware in a supply chain scenario is not possible because FIDO tokens are required to contain a so-called hardware attestation key. This key is used, together with the respective certificate signed by the manufacturer, to attest the token type and manufacturer during the registration at a service. The Nitrokey FIDO U2F hardware attestation key is stored in the secure element. Thus, exchanging the secure element without knowledge of this key is not possible without being detected. However, we uncovered a vulnerability in the I2C communication protocol which fits the adversarial model. The vulnerability allows adversaries to reprogram the

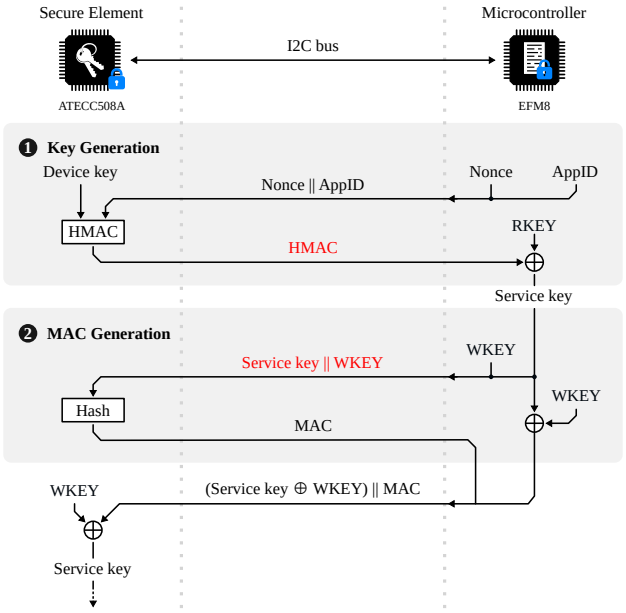


Figure 2: Part of the Nitrokey FIDO U2F registration flow. Messages on the I2C bus highlighted in red need to be eavesdropped by an attacker.

firmware of the microcontroller and add malicious features like low-entropy key generation or backdoor access to extract secret keys. This effectively compromises trust in the token since the attack can be carried out within minutes in either the supply chain or the evil maid scenario. Further security investigations using more elaborate attacks therefore seemed unnecessary.

5.1 Firmware Manipulation via Protocol Vulnerability

As with the secure element, the microcontroller cannot be simply replaced because it also contains secrets. These are stored in the internal flash memory. One of the two secrets, called WKEY, links both chips together and is also used to mask sensitive communication on the I2C bus. Without knowledge of this secret, the secure element cannot be used as an elliptic-curve cryptography (ECC) hardware accelerator. It is not an option to run the ECC operations in software instead, as it would be easily detectable during use due to the significant higher runtimes [GPW+04]. The other secret, called RKEY, is used to derive the service key for an authentication or registration on an U2F service. Since any kind of access to the flash memory via the debug interface of the microcontroller is restricted, these secrets can neither be read out nor can the firmware be manipulated. The only way to disable this protection is to erase the entire flash memory and thus both secrets.

However, due to the vulnerability in the I2C communication protocol, the secrets do not need to be extracted from the microcontroller. Instead, an attacker only needs to intercept the communication on the I2C bus with a logic analyzer during any (fake) registration to an arbitrary U2F service. This only requires to open the housing, which can be done without any tools and without leaving any traces as the casing is plugged together. The relevant messages that are exchanged during a registration are displayed in Figure 2 and are divided into two phases:

1. **Key Generation** An HMAC operation on the secure element is used as key derivation function (KDF). Therefore, the nonce and the AppID, both come from the FIDO client, typically a web browser, are transferred over the I2C bus to the secure

element. Afterwards, the microcontroller receives the resulting HMAC and XORs it with the RKEY to finally compute the service key.

2. **MAC Generation** In a later protocol stage a MAC is required in order to transfer the service key from the microcontroller to the secure element. Crucially, this MAC calculation is outsourced to the secure element. As a result, the microcontroller transfers the WKEY and the service key in plain to the secure element. Finally, the secure element sends the resulting MAC to the microcontroller. This is a surprising design choice and root cause of the vulnerability.

From Figure 2 one can see that an attacker only needs to eavesdrop two messages in order to recover WKEY and RKEY, these messages are highlighted in red. The first one is the result of the HMAC at the end of the key generation phase. The second one is the message that contains the inputs for the MAC calculation. This message contains the WKEY and service key in plain. The missing RKEY can be derived as an XOR between the HMAC and the service key.

Since the adversary is in possession of both secrets, the microcontroller can be erased and reprogrammed via the debug interface. The token firmware can be arbitrarily manipulated and may be returned to the user in an evil maid scenario or delivered in a supply chain attack scenario. The compromised token may, for example, exfiltrate secrets through covert channels such as hidden in protocol payload fields or generate weak cryptographic keys. Furthermore, in an evil maid scenario, the attacker can recover already registered service keys. In this scenario, however, the adversary must perform an authentication against the corresponding service to receive the nonce. The token uses the nonce to generate the service key on-demand. Therefore, the first authentication factor, which is usually a password, is required. Afterwards, the token can be cloned and returned to its owner. The cloned token allows the attacker to login on any service without being detected.

5.2 Countermeasures and Mitigation

The root cause of the identified vulnerability results from the externalization of the hash calculation to the secure element. We propose to calculate the hash function and the resulting MAC on the microcontroller instead. The current firmware occupies about half of the 40 KiB flash memory. Experimental results of hash functions implemented in software on similar 8-bit microcontrollers [Osv12] show that neither the required memory nor the speed performance justifies the externalization. This countermeasure prevents an adversary from reading the secret on the I2C bus and renders the presented attack impractical.

6 Nitrokey Pro 2

The Nitrokey Pro 2 is a multi-purpose security device that can serve as an OpenPGP smart card, password manager and second authentication factor. The device consists of an STM32F103 microcontroller from STMicroelectronics and a SmartMX P5CD081 security controller from NXP. As depicted in Figure 3, the microcontroller is placed on the top side and the security controller on the bottom side of the token's internal circuit board. Note that the security controller is not used for the second factor authentication feature which we concentrate on here. The Librem Key from Purism is identical to the Nitrokey Pro 2 and merely marketed with a different name [Nit18]. It is therefore also covered by this analysis. The Nitrokey Pro 2 supports authentication based on one-time passwords (OTPs). In particular, it can manage up to 15 time-based one-time password (TOTP) entries and four HMAC-based one-time password (HOTP) entries. To prevent unauthorized use without the user's consent, a PIN must be entered before OTPs can be generated. This unlock is valid for a certain period of time before the PIN must be entered

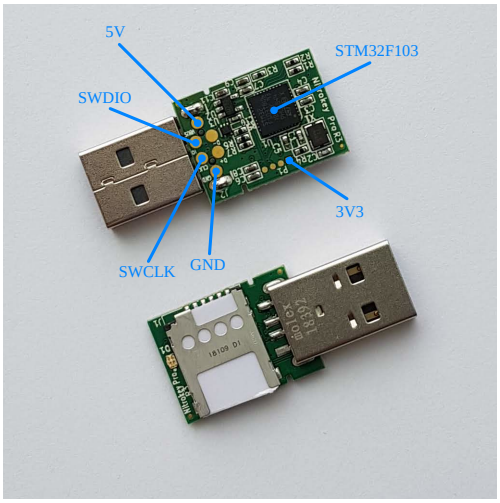


Figure 3: Front (top) and back side (bottom) of the Nitrokey Pro 2. The integrated microcontroller and the accessible supply and debug pins are highlighted.

again and protects against attackers with physical access to the token such as in an evil maid scenario. Even though a PIN is necessary to unlock the OTP functionality, all secrets are stored in plaintext in the microcontroller's flash memory. Hence, the confidentiality of the OTP secrets relies solely on the flash memory read-out protection features of the microcontroller.

We performed a security investigation according to our methodology and analyzed the architecture and included chips. We uncovered two attacks on the security token which both effectively compromise its trust in supply chain and evil maid scenarios. One is based on the fact that a memory protection feature is missing and another one is based on a vulnerability that has already been published as blog post [SO20] by one of the authors. Further security investigations using more elaborate attacks seemed unnecessary.

6.1 Firmware Manipulation in the Supply Chain

The STM32F1 microcontroller series does not provide a write protection and, thus, its flash memory can be erased and reprogrammed without limitation. Further the Nitrokey Pro 2 does not contain a hardware attestation key or comparable security measures, unlike the Nitrokey FIDO U2F as explained in Section 5. As a result, the token firmware can be easily manipulated in the supply and distribution chain. A user cannot detect any tampering during the supply chain and thus not trust the firmware on the token. Manipulated firmware may for instance integrate backdoor access to the secret keys stored on the microcontroller and may similarly affect the other security features of the device.

6.2 Key Extraction via Debug Interface Vulnerability

The flash memory read-out protection feature of the STM32F1 series exhibits several vulnerabilities [SO20, BFP19, OSM20]. The *Exception(al) Failure* vulnerability, that one of the authors already published on a blog [SO20], fits the adversarial model because it requires no modification of the device and, thus, leaves no evidence. Only physical access to the microcontroller's debug interface and a regular debug probe are required. Exploiting this vulnerability allows an adversary to extract large amounts of data from the microcontrollers' flash memory. Nitrokey confirmed [Suh20] this issue but does not

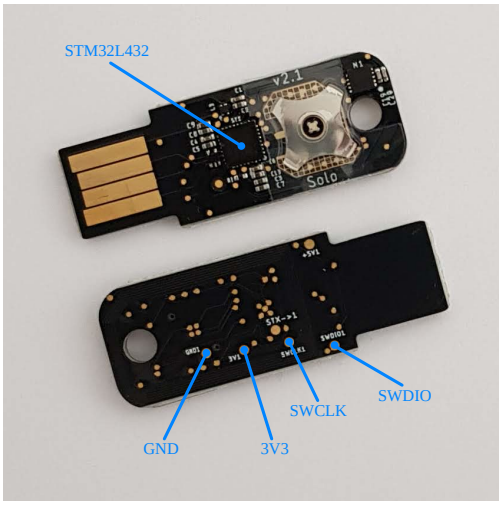


Figure 4: Front (top) and back side (bottom) of the SoloKeys Solo. The integrated microcontroller and the accessible supply and debug pins are highlighted.

provide any details of how it affects the Nitrokey Pro 2 and, especially, its application as second authentication factor.

Therefore, we hereby show the impact of the vulnerability on the OTP feature of the Nitrokey Pro 2. The plastic housing of the token can easily be opened and closed without leaving noticeable traces. Once opened, the debug interface is accessible from the top side of the circuit board, as indicated in Figure 3. By exploiting the vulnerability with the help of a debug probe [SO20], we gain access to the microcontroller's flash memory content. The flash memory contains secret keys and metadata, such as the service name, for each OTP entry. Most of this data can be successfully read out which means that the token can be maliciously cloned. However, due to the nature of the vulnerability, it is not possible to extract the entire flash memory. In total, the secret keys from 11 out of 19 OTP entries can be fully extracted. There are at least four unknown bytes in the other secret keys. Table 3 lists the number of bytes for each entry that cannot be extracted. While most of the OTP metadata can be retrieved via the regular USB interface, the counter values which are required for HOTP cannot be fully extracted. This means that HOTP keys cannot be used at all by the attacker.

An offline brute-force of unknown secret key bytes and counter values would require at least one respective valid OTP for a reference comparison. This reference value could only be obtained if either knowing the PIN or eavesdropping a transmission. We disregard both due to strong attacker assumptions. An online brute-force against the service would be impossible due to simple measures such as counting failed attempts.

In summary, our analysis shows that the OTP feature of the Nitrokey Pro 2 is vulnerable to physical attacks such as evil maid attacks due to a vulnerability in the used microcontroller. An adversary with physical access to the token is able to extract the secret keys from 9 out of 15 TOTP entries. With that, the adversary is able to (partially) clone the token or replace the firmware with a malicious one. The attack can be mounted within a couple of minutes and without leaving any evidence to the token owner.

6.3 Countermeasures and Mitigation

The Nitrokey Pro 2 has no protection against firmware manipulation at all. Hence, we suggest to build and program the firmware on-premise to hinder potential supply chain attacks and establish trust in the firmware running on the token.

As the used microcontroller has a weakness in its flash memory read-out protection, the content must be protected by other means. Since the user has to enter its PIN before being able to generate an OTP, we recommend to store the OTP secrets encrypted in the flash memory and derive a key from this PIN. This ensures the confidentiality of the OTP secrets without introducing an usability penalty. To prevent offline brute-force attacks against the encrypted flash content, a certain minimum PIN length is required. Also, a hash-based KDF can be used to force a higher computational effort per each tested PIN during brute-force.

7 SoloKeys Solo

A very popular security token is the Solo from SoloKeys. It implements the FIDO2 and the U2F specification. In contrast to the other two security tokens, the Solo only contains a general purpose microcontroller, an STM32L432 from STMicroelectronics. Figure 4 shows the front and back side of an opened Solo USB-A security token. The supply and debug pins are accessible from the back side. Since the Solo implements the FIDO2 and the U2F standard, its flash memory contains three valuable assets: the U2F master key, the FIDO2 resident keys and the hardware attestation key. The design of the Solo is used as basis for other open source security tokens such as the Somu from SoloKeys and the Nitrokey FIDO2 [Nit81]. Hence, our results also apply to these devices if not explicitly stated otherwise.

The security investigation of this token did not uncover any attacks on the architecture step. According to the methodology we then proceeded to analyze the cryptographic operations where we found a successful side-channel attack to extract secret ECC keys. After that, we investigated further and found a successful fault injection attack to circumvent a hardware security feature of the chip and extract the master key as well. The two attacks require more effort than the ones previously described, but are still within the adversarial model. They can be used to extract all credentials from the Solo which allows supply chain attacks, including deploying malware, as well as evil maid attacks. Note that simply replacing a token is not possible due to the integrated hardware attestation key, as discussed in Section 5.

7.1 Key Extraction via Side-Channel Analysis

The FIDO2 and U2F authentication protocols both require the elliptic curve digital signature algorithm (ECDSA) to be computed on the token. A side-channel vulnerability in the cryptographic operation would allow an attacker to recover the service and resident keys as well as the hardware attestation key from a token. The master key would not be affected since it is not used directly by the ECDSA operation itself. Since the STM32L4 does not provide a hardware accelerator for ECC, the cryptographic algorithms are implemented in software. The Solo uses the micro-ecc library [Mac11] which includes some protection against common side-channel attacks. It uses the Montgomery Ladder for a (theoretically) constant execution flow during elliptic curve scalar multiplications to prevent SPA and some forms of horizontal attacks. As pointed out in a study on ECC libraries [Sil19], the micro-ecc library does not randomize the base point in its ECDSA implementation.

The implementation includes countermeasures but an uncovered flaw allows an attack which requires comparably low effort to extract the secrets used to generate signatures during an authentication or registration process. The attack is novel to the best of our knowledge but can and should be prevented as described below. Further attacks, such as horizontal attacks which exploit leakage from secret dependent register (re-)usage are possible, but were not investigated further. Note that some horizontal attacks require

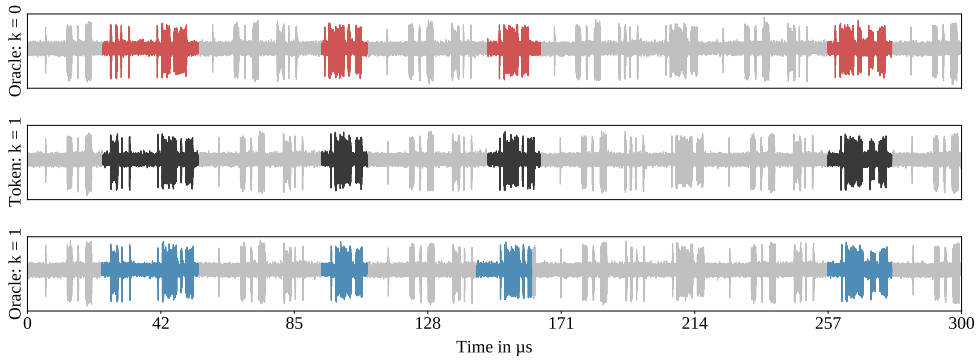


Figure 5: EM traces of the first ladder step of the attacked token (middle) and the oracle with k set to 0 (top) and 1 (bottom). Regions where distinctive patterns from the oracle trace match with the token are highlighted in blue and red otherwise.

high-precision measurements to exploit smaller sources of leakage, however, such equipment is outside the adversarial model.

Oracle-based Single Trace Attack We attack the scalar multiplication $k \times G$ which is the core operation of ECDSA signatures. Through low-effort initial side-channel measurements, we found that the instruction flow and timing of each Montgomery ladder step in fact differs significantly for varying inputs. Even though the higher-level Montgomery algorithm is constant-time, there seem to be subfunctions with data-dependent runtimes. This design flaw is sufficient for a successful single trace SPA-style attack which is also similar to horizontal attacks in classifying subsequent secret bit-dependent iterations and uses an external oracle to predict variations.

A single signature generation is recorded of the device using a LeCroy WavePro 7 Zi-A 2.5 GHz oscilloscope operating at a sampling rate of 100 MS/s and a Langer RF-R 50-1 near field probe. This fits the adversarial model, but even a hand-wound coil can be used as a low-cost probe. The probe is centered on top of the microcontroller package as depicted in Figure 10a. All remaining steps can be done after returning the token.

The main idea of the attack is to use a second STM32L4 device as *leakage oracle*: We program it with the same micro-ecc library as it is used on the attacked device, but modify the software such that we can set the inputs to the Montgomery Ladder steps. Since the base point is not randomized, the only unknown input to the first ladder step is the first bit of k. We configure the initial state of the Montgomery Ladder on our leakage oracle with the base point and measure two executions of the first ladder step: one where the first bit of k is 0 and one where it is 1. Through visual comparison of the patterns in the EM traces of the oracle to the attacked trace, it is easy to identify which bit value is correct. In Figure 5, an example for the first ladder step is depicted. It shows the EM trace of the attacked token in the middle, the oracle traces for $k = 0$ on the top and for $k = 1$ on the bottom. Regions where distinctive patterns from the oracle trace match with the attacked device are highlighted in blue, and marked in red when not matching. By comparison one can clearly see that the token processes the ladder step with $k = 1$. For all remaining bits of k, the oracle is repeatedly updated with the known bits and then the respectively next ladder step is computed for the two hypothetical values of the next bit. Due to the length of a ladder step and the large number of data dependencies in the code this led to unambiguous results in all our experiments. Hence, we are able to fully recover k from a single observation on the attacked device and extract the hardware attestation

Table 2: Read-out protection (RDP) levels of the STM32L4 microcontroller series.

RDP level	Option byte value	Behaviour
0	0xAA	Full debug access
1	any other	Limited debug access
2	0xCC	No debug access

key, the FIDO2 resident keys or the U2F service keys. For the U2F service keys the same applies as for the vulnerability of the Nitrokey FIDO U2F token: to determine already registered service keys the first authentication factor must be known, which is usually a password for a web service.

The described attack requires recording a single trace of the device and some manual effort afterwards, but could be automated further. This is realistic in a supply chain scenario but also feasible in an evil maid scenario. In a supply chain scenario, it allows an adversary to extract the private hardware attestation key and, with it, to create authentic but possibly manipulated tokens as explained in Section 5. Such tokens could for example generate weak service keys with low entropy which would be a severe security risk, especially when using passwordless logins.

7.2 Key Extraction via Fault Injection

The previous side-channel attack recovers the hardware attestation key, the FIDO2 resident keys and the U2F service keys but not the U2F master key which would allow an attacker to derive all past and future service keys immediately. Hence, we performed further investigations following our methodology which means that we investigated the used security features of the chip.

We found a way to extract secrets by circumventing the read-out protection of the STM32L4 microcontroller. This protection originally ensures the confidentiality of the flash memory and prevents unauthorized read-outs via the debug interface. The STM32L4 series comprises three read-out protection (RDP) levels that are listed in Table 2. In level 0, no security is enabled and the flash memory is accessible via the debug interface. In level 1, the debug interface is still open but the access to the flash memory is restricted. Additionally, once a debug probe is attached, the firmware execution stops and cannot be resumed until the microcontroller is power-cycled. A downgrade from level 1 to 0 is always possible but results in a full erase of the flash memory. In level 2, the debug interface is completely and permanently deactivated.

As documented in the reference manual, the RDP level is part of the option bytes. Each option byte is stored together with its complement in flash memory. This redundancy is meant to detect errors either during an erroneous transfer or caused by bit errors in the flash memory. The option byte values for the respective RDP levels are listed in Table 2. This mapping has been identified as vulnerable to fault injection attacks in several publications [OT17, RDN19, BFP19]. From Table 2, it is obvious that with this encoding a single bit flip is sufficient to downgrade the RDP level from 2 to 1. However, none of the already published attacks are applicable to our target and attack scenario. The reason for this is that these attacks require time consuming preparation or even invasive measures such as chip decapsulation.

Downgrade to Level 1 During the boot phase and before the firmware is executed the five 64-bit wide option byte sections are loaded from flash memory into internal registers which are not accessible to the user. Afterwards, the option bytes are transferred to the

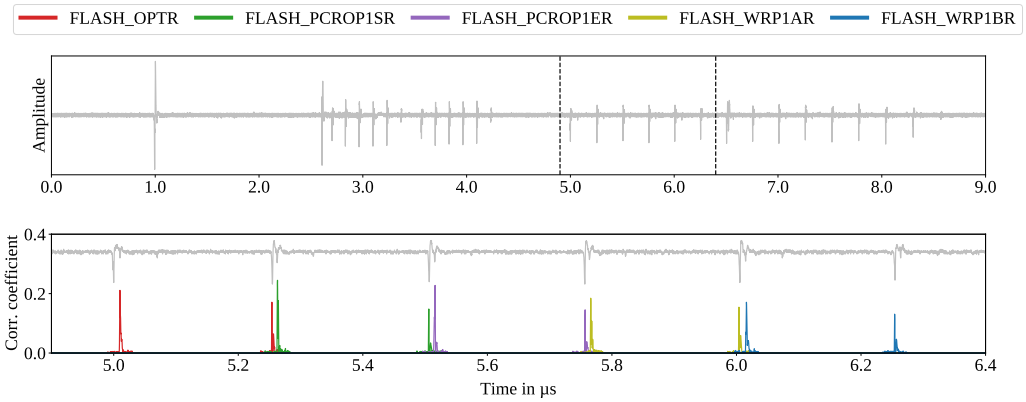


Figure 6: EM trace during the boot phase of an STM32L4 microcontroller (top). The zoom (bottom) of the region between the two dashed lines shows the results of a correlation-based leakage test on the option byte registers with 200 000 traces.

five 32-bit wide user accessible option byte registers. In case of a mismatch between option bytes and their complements, a default value is used instead. For the RDP level, the value `0xFF` is used, which is surprising as it maps to level 1 [STM17]. Hence, the actual vulnerability that allows a downgrade from level 2 to 1 is not the RDP level encoding. Instead, it is the default RDP value that is used in case of a mismatch between the option bytes and their complements.

The Solo is shipped in RDP level 2, so according to the specification the debug interface is permanently disabled and a level downgrade is not possible. However, to reactivate the debug interface nonetheless, we inject a glitch during the boot phase to cause an erroneous memory transfer of the `FLASH_OPTR` option bytes. As described, the microcontroller then falls back to RDP level 1. The precise timing is crucial for a successful glitch. The attacker needs to glitch the transfer of the option bytes from flash memory to the internal registers. We use an EM side-channel recording of the microcontroller during the boot phase to learn the precise timing. For the acquisition we use a LeCroy WavePro 7 Zi-A 2.5 GHz oscilloscope with a sample rate of 1 GS/s, a Langer RF-U 2.5-2 near field probe and a Langer PA 303 preamplifier adding a gain of 30 dB. As an alternative, the 4 mm injection tips of the ChipSHOUTER can be used as low-cost replacement for the EM probe. The probe is manually positioned on the chip package so that clearly visible peaks are recorded. In Figure 10b, the final position of the probe is shown. The EM signal of the boot phase is depicted in the top of Figure 6. Multiple peaks can be observed in the EM signal that may correspond to bus transfers. To determine whether these peaks correspond to the option byte registers, we make use of a common method in side-channel analysis (SCA): the detection of leakage and points of interest. We specifically use the correlation-based leakage test described by Durvaux and Standaert [DS16]. We perform this profiling using another, unprotected STM32L4 microcontroller. For the leakage test we record 200 000 traces with different random values as option bytes.

The leakage test shows significant correlations in the region between the two dashed lines in the upper plot of Figure 6. The bottom plot is a magnification of this region with the correlation results of the leakage test. The leakage test clearly shows a correlation of approximately 0.2 for each option byte register. For the sake of clarity we show only the correlation of a single byte of each register. With the help of this leakage test, we identify the point in time at which each option byte is transferred from flash memory into the internal register. Thereby, we narrow down the search space for a successful glitch from the entire boot time of approximately 1.2 ms to an interval of around 200 ns. In case

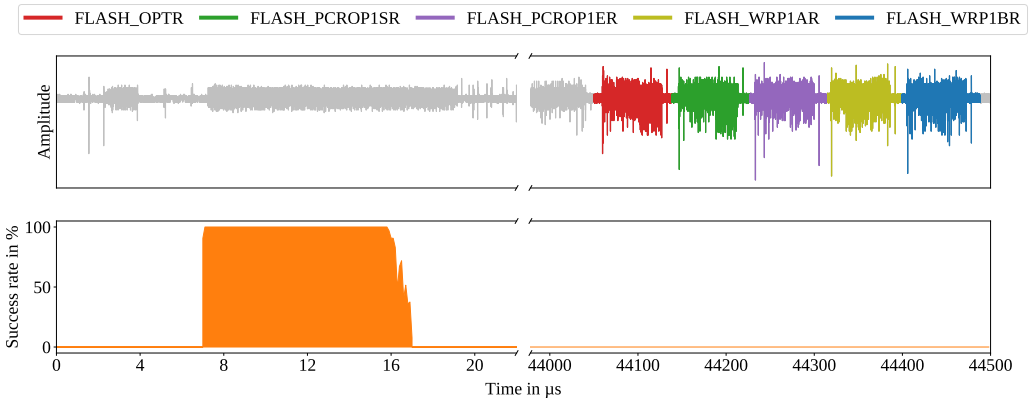


Figure 7: EM trace (top) and success rate of an injected glitch (bottom) during an RDP downgrade to level 0. The write operations of the option bytes are highlighted in color.

of the `FLASH_OPTR`, that is at around 5 µs. Note that the point in time may vary between different devices but the pattern in the EM trace remains the same and can be utilized as trigger for the fault injection. This approach is of major importance when access to the target token is limited in time like in an evil maid scenario.

We use electromagnetic fault injection (EMFI) to disturb the memory transfer of the `FLASH_OPTR` register and successfully downgrade the device to level 1. EMFI does not leave any traces on the target device and no preparation or decapsulation is required. Specifically, we use the ChipSHOUTER from NewAE Technology with the clockwise (CW) version of the 4 mm injection probe. The device is configured to a coil voltage of 500 V and a pulse width of 80 ns. The coil is positioned at the top left corner of the chip package as schematically outlined in Figure 10b. The distance between chip package and coil is about 330 µm or the thickness of one sheet of photo paper. With this configuration, we achieve an attack success rate of 100 %. Our assumption that the downgrade is caused by the fallback of the RDP level is confirmed by the fact that the `OPTVERR` bit in the flash status register (`FLASH_SR`) is set after the attack. This bit indicates that the option byte values and their complements do not match [STM17]. Note that the STM32L4 seems not to be susceptible to voltage glitches. Using a ChipWhisperer voltage glitching device with the exact same trigger setup, we were unable to execute a similar downgrade.

Key Extraction from SRAM After the successful downgrade to level 1, the debug interface is unlocked and access to the CPU registers and SRAM is granted, but not to the flash memory. Interestingly, the firmware loads the U2F master key into the SRAM right after the start. Hence, an attacker can easily extract it using a debugger and is thereby able to derive any existing or future service key generated by this token. This is a complete break of the device.

The FIDO2 resident keys and the attestation key reside in SRAM only when they are used during authentications to a service and initial registration, respectively. The attestation key can be extracted from SRAM by triggering a registration via the USB interface, during which it is used by the function `uECC_sign_with_k()` of the micro-ecc library. The CPU can be halted right after the key is loaded from flash into SRAM by using the reset signal. The timing and a useful trigger condition need to be found. Every registration requires the user to press the built-in button for start and confirmation after which an LED lights up. This can be used as a trigger. The timing is recovered by running the Solo firmware on a second device after including a signal close to the respective key load operation. Alternatively, a simple trial and error approach would be feasible. The

steps to extract the attestation key from the Solo *after* the RDP level downgrade are:

1. Power-cycle the microcontroller to start firmware execution in RDP level 1. During the initialization phase, the firmware changes the level back to 2 by writing the respective option bytes into the flash memory. However, level 1 remains active until the next option byte loading or power-cycle is performed.

2. Start a FIDO2 registration process and press the user button on the device. This loads the attestation key into the SRAM in order to perform the cryptographic operations.

3. After a delay of about 275 ms, pull the reset line to halt the processor. The microcontroller performs a reset but the SRAM content is preserved.

4. Connect to the microcontroller using a debugger and read out the secret attestation key from SRAM. Optionally, set the RDP level to 1 for a subsequent attack after power-cycling.

The described attack takes less than a minute and after learning the timing as described, the success rate is 100 %. Note that the timing may vary between different tokens but the attack can be fully automated to compensate slight differences.

In summary, all secrets of the Solo can be extracted using low to moderate attacker efforts, if no user PIN is used. The costs of the required equipment are in the range of 5000 USD and a further reduction by using a custom-made EMFI setup is likely. This means that the trust of the token can be broken by attacks within the specified adversarial model in both supply chain as well as evil maid scenarios.

Downgrade to Level 0 If the token is configured with a user PIN, credentials cannot be recovered from SRAM. Instead, they must be extracted from flash memory which is only accessible in RDP level 0. However, a downgrade to level 0 by glitching the memory transfer seems infeasible because the option byte for the RDP level and its complement byte would need to be set to the specific values `0xAA` and `0x55`, respectively. Instead, we describe how a regular downgrade from level 1 to 0, which should mass erase the flash memory, can be tricked and successfully exploited to retain its content.

The EM side-channel trace during the downgrade from level 1 to 0 is depicted on the top of Figure 7. Our analysis hints that this downgrade consists of three phases: (1) an option bytes erasure, (2) a full mass erase and (3) an option bytes write. We found that the mass erase is susceptible to EMFI glitches. With that, we can skip the mass erase without disturbing the following write operation of the option bytes. As a result, we are able to successfully change the RDP level from 1 to 0 without a mass erase. Afterwards, the entire flash memory can be read out via the debug interface. The positions of the EM probe and EM injection coil for this attack are schematically outlined in Figure 10c.

In a real-world attack, the reliability of this method would be crucial because credentials on the token would be irretrievably lost otherwise. The experiment exhibits a 100 % success rate when the glitch is injected within a certain time interval of approximately 8 µs. This time interval is highlighted in the lower part of Figure 7. The figure also includes the success rate for 64 fault injections at the respective individual times.

In contrast to the previous attacks which were carried out on the actual security tokens, we evaluated this one on a breakout board in order to be more adjustable with the EMFI coil position. Instead of the STM32L432 which is integrated in the Solo, we used an STM32L422 because it is available in a package that is more comfortable to solder. The behaviour during the downgrade from level 1 to 0, as depicted in Figure 7, is the same on both microcontrollers. However, due to the different hardware, for example the chip package, this attack cannot directly be applied to the Solo. Chip specific parameters, such

as the EM coil position, must be determined first. We include these results nevertheless to demonstrate that a full break of the read-out protection is possible and to emphasize the need for appropriate countermeasures.

7.3 Countermeasures and Mitigation

In order to prevent the side-channel attack on the micro-ecc library, we suggest to use randomization of the projective coordinates in the ECDSA implementation, since the code uses a Montgomery ladder and a projective coordinate algorithm. Alternatively, the base point could be randomized as proposed in a study [Sil19] about ECC libraries. Either measure would effectively prevent the specifically described SCA. Randomization countermeasures will generally prevent any profiling or prediction of intermediate values for single trace, SPA and horizontal attacks. Data-dependent instruction timing or operand usage may, however, still be exploitable using non-profiled horizontal attacks [HIM+13, NC17]. Most such attacks require high-precision equipment to exploit small leakage sources, which is out of scope for our adversarial model [HIM+13]. Nonetheless, randomization of operations and storage locations can be used to mitigate remaining leakage sources as summarized by Nascimento and Chmielewski [NC17].

To protect against the flawed read-out protection, we recommend that the master key is loaded into the SRAM only when needed and not at the beginning of the firmware execution. It should also not be stored there permanently. Alternatively, the dedicated SRAM2 can be used as key storage. In RDP level 1, this memory region is inaccessible via the debug interface. In addition, we suggest to improve the present option byte verification during the device initialization. Currently, the firmware writes the option bytes with the highest protection level during the device startup but does not immediately activate these written values if an unexpected lower level is active. This allows the attacker to downgrade the RDP level from 2 to 1 using a fault attack and extract the SRAM content afterwards. We propose to first compare the option byte register to the desired level and, in case of a discrepancy, to overwrite and explicitly activate it. This should be performed immediately during the device initialization such that there is no possibility for the attacker to exploit faulted option bytes.

Additionally, the token should enforce a PIN entry for all functionality. This PIN should not only be checked by the token, but also used to encrypt the credentials stored in flash memory. This would help against complete fails of the read-out protection as outlined in the downgrade attack to RDP level 0. Again, the PIN would have to contain enough entropy to prevent offline brute-force attacks. In contrast to the Nitrokey Pro 2 where the PIN is already mandatory, the additional PIN introduces a trade-off between security and usability.

8 Google OpenSK

OpenSK is a research platform initiated by Google that aims to provide an open source firmware for security tokens. The firmware is written in Rust and supports both the U2F and the FIDO2 standards [Ope73]. The FEITIAN OpenSK Dongle [FEI21] is a commercially available token that is based on this platform.

Since the current project supports only the nRF52840 from Nordic Semiconductor as a target microcontroller, we examine this device in the following. In contrast to the microcontrollers used in the other tokens, it features a hardware accelerator for cryptographic operations, namely the ARM CryptoCell 310 [Nor18].

The security investigation of this token is quite similar to the Solo as no attacks on the architecture step were uncovered. From there onwards to the cryptographic operations, a successful side-channel attack was found to extract secret ECC keys. Again, we have

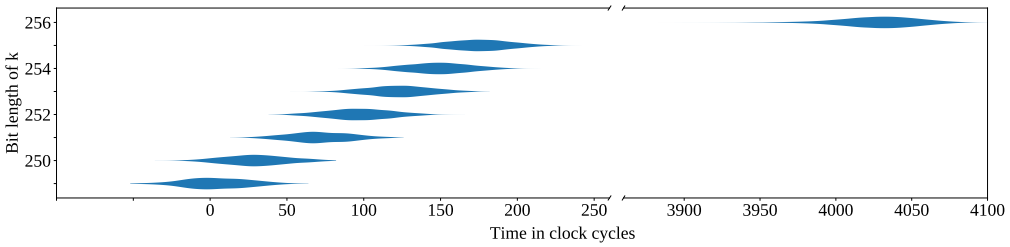


Figure 8: Relative time variation of signature generation depending on length of k.

investigated further into the hardware, which also led to the discovery of a successful fault injection attack to circumvent the hardware security features. The vulnerabilities allow an attacker to compromise the OpenSK in a supply chain as well as in an evil maid scenario. As with the Solo, simple hardware replacements are hindered by the integrated hardware attestation key, as argued in Section 5.

As the OpenSK Dongle was not available for purchase in Europe at the time of writing, we carried out the attacks on an nRF52840 development kit. It contains the same chip with the same package as used by the OpenSK Dongle. According to the token's hardware design files and pictures, the coil placements are not hindered by any components. Also, the debug interface pins are accessible from the top side of the token's circuit board. For that reason, we are certain that the attacks are reproducible on the OpenSK Dongle.

8.1 Key Extraction via Side-Channel Analysis

To compute ECDSA signatures, the project currently uses a software library but the documentation states that the ECC hardware accelerator will be used in the future [Ope73]. To assess the security of this accelerator, we performed a side-channel analysis and observed a non-constant time signature computation. The execution time correlates with the bit length of the secret scalar k. This correlation is obvious in Figure 8 where the observed execution time is depicted on the x-axis and the bit length is depicted on the y-axis.

In recent years, several vulnerabilities have been found in hard- and software ECC implementations which have been used to extract the private key [MSEH20, ECTe9]. We employ a lattice attack by using the open source tools from Jancar et al. [JSSS20] which successfully recovers the secret scalar k based on the observed timing leakage of the nRF52840. Further, we investigate how many observed executions are required for a successful attack. For that, one million signatures are computed and the execution time is recorded using the on-chip SysTick timer. The measurements are split into randomized subsets with a quantity of 200 to 20000 signatures with a step size of 200. For each set quantity, the random selection and attack is repeated 24 times. The resulting success rate for a fully recovered secret is plotted in Figure 9. It can be observed that around 12000 signatures are sufficient for secret recovery. This is a feasible amount if the attacker has possession of the key for half a day, for instance. The described attack is completely non-invasive and requires no modifications of the hardware. Attackers may exploit this vulnerability to recover the hardware attestation key in a supply chain scenario. This enables arbitrary manipulations of the firmware, such as generation of weak cryptographic keys, without being noticed by the token owner. Attackers may also recover resident keys in an evil maid scenario. However, the U2F master key cannot be recovered, but instead already registered U2F service keys as stated in Sections 5 and 7.1.

The on-chip accelerator is proprietary hardware provided by ARM and comes together with a closed source software library for interfacing. Since both components are closed source, the root cause of the vulnerability cannot finally be identified in this work.

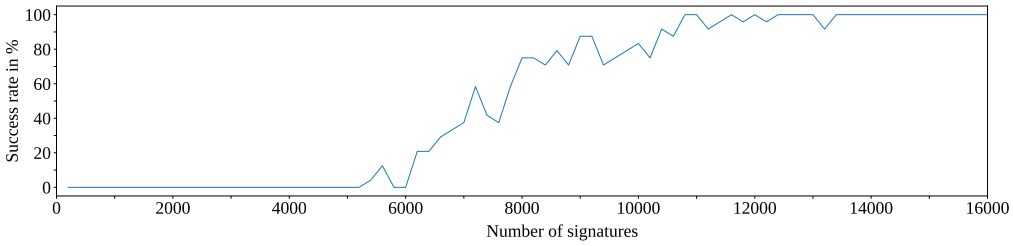


Figure 9: Success rate to recover an ECC key depending on number of signatures.

8.2 Key Extraction via Fault Injection

The nRF52840 features a flash read-out protection that deactivates the debug interface to the core and all peripherals. A reactivation is only possible when performing a mass erase of the RAM and flash memory [Nor18]. However, by desoldering two decoupling capacitors and injecting a voltage glitch during the boot phase of the microcontroller this read-out protection feature can be circumvented [Lim20]. In case of a supply chain attack, this allows to extract the private hardware attestation key and manipulate the firmware of such tokens. For an evil maid attack, however, the desoldering is inconvenient because it likely leaves traces, which does not fit our attacker model. We investigate whether the described attack can be carried out using EMFI without leaving any trace and thus proof that a fully non-invasive attack on this device is possible. The same trigger setup as for the voltage glitch and the same EMFI setup as for the attack on the Solo is used. The coil position of the setup is schematically outlined in Figure 10d and the distance to the chip package was about 330 µm. We are able to reactivate the debug interface with a success rate of about 30 % for a single attempt. The fault injection can be repeated until success is achieved. The entire attack can be carried out in less than a minute. In contrast to the attack described in Section 7.2, this one is even more severe because a single successful attack gives full access to the flash memory. Based on this finding, we state that an evil maid attack is feasible against the microcontroller and a respective security token. Similar to the case of the Solo, this attack allows to extract all credentials and to manipulate the token's firmware without leaving any evidence.

8.3 Countermeasures and Mitigation

The hardware accelerator for ECC of the ARM CryptoCell 310, which is integrated on the nRF52480 microcontroller, exhibits a timing side-channel vulnerability. Even though this side-channel could probably be mitigated on the application layer, the leakage is highly likely also exploitable via other side-channels such as power or EM. Hence, we suggest to use an open source cryptographic library that includes the countermeasures we recommended for the micro-ecc library. In order to deal with the flawed read-out protection of the nRF52840, we suggest to enforce a user PIN and store the secrets encrypted in flash memory, as discussed for the Nitrokey Pro 2 and Solo in Sections 6 and 7, respectively.

9 Recommendations for Future Security Tokens

In this section, we present important insights and recommendations for designers and manufacturers of future security tokens.

General Purpose Microcontrollers We do recommend to use a single general purpose microcontroller for security tokens since this most likely enables trustworthy and secure designs. The main reason for choosing a general purpose microcontroller is that the entire firmware can be published as open source to establish trust by public review. However the particular choice of a microcontroller is important, because their specific security features and properties are decisive.

Open Source Hardware Designs Security features which are integrated in microcontrollers are usually closed source, implementation details are only known to chip manufacturers and cannot be reviewed independently by third parties. Designers of tokens must therefore trust chip manufacturers, which is not always justified as the results of this contribution clearly show. The topic of open source hardware designs is gaining significant momentum currently and we highly recommend to use open source microcontrollers in the future. Unfortunately, no mature products are available commercially yet.

Debug Interface Protection Microcontrollers typically include a debug interface which enables unrestricted privileged access to its core and all peripherals. As the single most important and essential hardware feature, a *debug interface protection* is required which restricts read and write access to internal peripherals and especially the flash memory. However, sometimes there are complex dependencies with other features, safety mechanisms and even liability requirements. This leads to complex designs and implementations which are inherently prone to errors. This is evident in our findings regarding the EMFI attacks in Sections 7.2 and 8.2. We emphasize that for secure designs, the activation of protection levels should be a one-way mechanism. An integral part is to retain the protection state irreversibly in non-volatile memory, for which electronic fuses (eFuses) can be an option. In comparison to flash memory cells, they are advantageous because they are not susceptible to semi-invasive attacks with ultraviolet (UV) light as presented in previous publications [Hua, OT17]. Further, it is important how to process the protection state information. At least, the highest available protection level should provide a means to reliably make the protection level permanent without unprotected and involuntary downgrading mechanisms. In other words, an irreversible activation of a rigorous debug interface protection is required for secure designs. This feature is essential to prevent attackers with physical access from reading out cryptographic secrets.

Unfortunately, debug interface protection features have proven flawed many times. Also, they cannot prevent more sophisticated attacks such as invasive chip probing. For these reasons, we strongly suggest to store credentials in an encrypted form. Since the FIDO2 specification already supports the use of a PIN or biometric mechanism to unlock the security token, the same can be used to decrypt the credentials on-demand. Note that the PIN length and a KDF for increased computational complexity need to be chosen carefully to protect against brute-force attacks.

Isolated Key Storage Even with an effective debug interface protection, cryptographic secrets can generally be vulnerable to software-based attacks (which are not specifically considered in this contribution). Since the firmware inevitably needs access to such credentials to perform the specified cryptographic operations, data leakage bugs due to missing or incorrect bounds checks are possible. One way to limit this is to use an integrated memory protection unit (MPU) which helps to reduce the code that is able to access credentials to a bare minimum. However, an adversary may gain privileged access and disable the MPU. To prevent all software-based attacks we would like to propose a feature for future microcontrollers with the following properties: (1) a non-volatile memory to store cryptographic keys, (2) an interface to securely generate keys and (3) an interface for cryptographic algorithm implementations to access such keys without the help of the

firmware. No such products are currently available and we encourage manufacturers to include such a feature in future products. Similar properties can be achieved by using a microcontroller which includes a trusted execution environment for the isolation of cryptographic operations and keys.

Cryptographic Implementations Cryptographic algorithms are required for the authentication functionality and other purposes such as secure firmware updates. It is crucial that cryptographic implementations are protected against fundamental side-channel analysis according to the described adversarial model. For instance, by ensuring data-independent execution times and utilizing state of the art blinding and masking techniques. The side-channel attacks in Section 7.1 and Section 8.1 demonstrate the consequences of vulnerable cryptographic implementations. When using hardware implementations, microcontroller products shall be selected accordingly. However, since performance is not crucial for security tokens, software libraries can be used instead which can even be updated for increased protection at later stages of development.

Firmware Integrity and Attestation Users may gain some trust in an acquired security token by visual inspection of the hardware, for instance the integrated microcontroller. The firmware, however, remains a black box to the user unless it is flashed by themselves. We would like to propose that future microcontrollers should include firmware integrity checking and attestation features which are implemented in hardware. They would allow end users to verify that the firmware has not been manipulated at any point during the supply chain. Unfortunately, this is not yet available and subject of future work.

10 Conclusion

In this work we show that current open source security tokens are susceptible for realistic adversaries with physical access in supply chain and evil maid scenarios. Especially supply chain adversaries are a major issue in our opinion because there is no way for end users to distinguish whether an acquired product is trustworthy or if it has been manipulated. Fortunately, we are able to contribute to the respective open source projects by proposing firmware modifications which mitigate all uncovered attacks. In fact, most of the identified vulnerabilities stem from the integrated microcontrollers. We cautiously expect more vulnerabilities in such devices due to their closed source nature. While certified products likely provide higher security at the moment due to the high efforts spent by manufacturers, our contribution serves as an example of how the quality of open source products effectively improves through independent research. Since the entry barriers are low, research teams from all over the world may easily contribute over time. More certification is not the best solution in our opinion since attacks on certified devices have also been demonstrated repeatedly and mitigation requires design cycles with huge efforts. We think that only open source hardware designs, that can be reviewed and advanced by a world-wide community, will be able to support secure and trustworthy products in the long term. No single manufacturer will be able to spend as many resources. As a result, we look forward to open source hardware designs and microcontrollers. As a specific shortcoming of current microcontroller products, we would like to highlight that hardware-based concepts for effective and lightweight attestation of firmware trustworthiness are missing.

Acknowledgements The work presented in this contribution was funded by the German Federal Office for Information Security (BSI). We thank Sven Freud and Tobias Damm for the valuable discussions and feedback.

References

[BFP19] Claudio Bozzato, Riccardo Focardi, and Francesco Palmarini. Shaping the Glitch: Optimizing Voltage Fault Injection Attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):199–224, February 2019.

[Bis19] Paul Bischoff. Report: 267 million Facebook users IDs and phone numbers exposed online. https://www.comparitech.com/blog/information-security/267-million-phone-numbers-exposed-online/, December 2019.

[Cor18] Microsoft Corporation. Password-less protection. Technical report, December 2018.

[Cox14] Sean Gallagher Cox. Experts Call for Transparency Around Google's Chinese-Made Security Keys. https://arstechnica.com/tech-policy/2014/05/photos-of-an-nsa-upgrade-factory-show-cisco-router-getting-implant/, May 2014.

[CS20] Simon Collin and Francois-Xavier Standaert. Side channel attacks against the Solo key - HMAC-SHA256 scheme. 2020.

[DCM+19] Emma Dauterman, Henry Corrigan-Gibbs, David Mazières, Dan Boneh, and Dominic Rizzo. True2F: Backdoor-Resistant Authentication Tokens. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 398–416. IEEE, 2019.

[DS16] François Durvaux and François-Xavier Standaert. From Improved Leakage Detection to the Detection of Points of Interests in Leakage Traces. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 240–262. Springer, 2016.

[ECTe9] ECTester. https://github.com/crocs-muni/ECTester, Commit: 31747e9.

[FEI21] FEITIAN. OpenSK USB Dongle V2. https://ftsafe.us/products/opensk-v2, 2021.

[Gib16] Samuel Gibbs. Dropbox hack leads to leaking of 68m user passwords on the internet. https://www.theguardian.com/technology/2016/aug/31/dropbox-hack-passwords-68m-data-breach, August 2016.

[GPW+04] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In *International workshop on cryptographic hardware and embedded systems*, pages 119–132. Springer, 2004.

[HIM+13] Johann Heyszl, Andreas Ibing, Stefan Mangard, Fabrizio De Santis, and Georg Sigl. Clustering Algorithms for Non-Profiled Single-Execution Attacks on Exponentiations. In *International Conference on Smart Card Research and Advanced Applications*, pages 79–93. Springer, 2013.

[Hua] Andrew Huang. Hacking the PIC 18F1320. https://www.bunniestudios.com/blog/?page_id=40.

[Ike20] Scott Ikeda. 250 Million Microsoft Customer Service Records Exposed; Exactly How Bad Was It? https://www.cpomagazine.com/cyber-security/250-million-microsoft-customer-service-records-exposed-exactly-how-bad-was-it/, February 2020.

[Inc17] Verizon Communications Inc. 2017 Data Breach Investigations Report: Executive Summary. Technical report, April 2017.

[JSSS20] Jan Jancar, Vladimir Sedlacek, Petr Svenda, and Marek Sys. Minerva: The curse of ECDSA nonces (Systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces). *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):281–308, 2020.

[Kre18] Brian Krebs. Google: Security Keys Neutralized Employee Phishing. https://krebsonsecurity.com/2018/07/google-security-keys-neutralized-employee-phishing/, July 2018.

[Lim20] LimitedResults. nRF52 Debug Resurrection (APPROTECT Bypass) Part 1. https://limitedresults.com/2020/06/nrf52-debug-resurrection-approtect-bypass/, June 2020.

[Mac11] Ken MacKay. micro-ecc. https://github.com/kmackay/micro-ecc, Commit: 601bd11.

[MSEH20] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, and Nadia Heninger. TPM-FAIL: TPM meets Timing and Lattice Attacks. In *29th USENIX Security Symposium (USENIX Security 20)*, Boston, MA, August 2020. USENIX Association.

[NC17] Erick Nascimento and Łukasz Chmielewski. Applying Horizontal Clustering Side-Channel Attacks on Embedded ECC Implementations. In *International Conference on Smart Card Research and Advanced Applications*, pages 213–231. Springer, 2017.

[Nit] Nitrokey. https://www.nitrokey.com/about.

[Nit18] Nitrokey. Nitrokey Partners with Purism to Build the Librem Key. https://www.nitrokey.com/news/2018/nitrokey-partners-purism-build-librem-key, October 2018.

[Nit81] Nitrokey FIDO2. https://github.com/Nitrokey/nitrokey-fido2-firmware, Commit: 8042681.

[Nor18] Nordic Semiconductor. *nRF52840 Product Specification*, March 2018. Rev. 1.0.

[O'F19] Colin O'Flynn. MIN()imum Failure: EMFI Attacks against USB Stacks. In *13th USENIX Workshop on Offensive Technologies (WOOT 19)*, Santa Clara, CA, August 2019. USENIX Association.

[Ope73] OpenSK. https://github.com/google/OpenSK, Commit: b0c1b73.

[OSM20] Johannes Obermaier, Marc Schink, and Kosma Moczek. One Exploit to Rule them All? On the Security of Drop-in Replacement and Counterfeit Microcontrollers. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, August 2020.

[Osv12] Dag Arne Osvik. Fast Embedded Software Hashing. Cryptology ePrint Archive, Report 2012/156, 2012. https://eprint.iacr.org/2012/156.

[OT17] Johannes Obermaier and Stefan Tatschner. Shedding too much Light on a Microcontroller's Firmware Protection. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC, August 2017. USENIX Association.

[Pre20] Associated Press. Official: Puerto Rico Govt Loses $2.6M in Phishing Scam. https://www.nytimes.com/2020/02/13/us/puerto-rico-phishing.html, February 2020.

[RDN19] Thomas Roth, Josh Datko, and Dmitry Nedospasov. chip.fail: Glitching the silicon of the Internet-of-Things. https://chip.fail/, August 2019.

[Roc21] Victor Lomneand Thomas Roche. A Side Journey to Titan: Side-Channel Attack on the Google Titan Security Key. Technical report, January 2021.

[RP13] David Oswald Bastian Richter and Christof Paar. Side-Channel Attacks on the Yubikey 2 One-Time Password Generator. In Salvatore J. Stolfo, Angelos Stavrou, and Charles V. Wright, editors, *Research in Attacks, Intrusions, and Defenses*, pages 204–222, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[SG17] Steve Scherer and Massimiliano Di Giorgio. Italy arrests two for hacking into emails of ECB's Draghi, former Italy PM Renzi. https://www.reuters.com/article/us-italy-cybercrime-idUSKBN14U1K2, January 2017.

[Sil19] Tjerand Silde. Comparative Study of ECC Libraries for Embedded Devices. Technical report, Norwegian University of Science and Technology, March 2019.

[SO20] Marc Schink and Johannes Obermaier. Exception(al) Failure – Breaking the STM32F1 Read-Out Protection. https://blog.zapb.de/stm32f1-exceptional-failure/, March 2020.

[Sol] SoloKeys. https://solokeys.com/.

[Sol20] SoloKeys. Security Analysis of the Solo Firmware. https://solokeys.com/de/blogs/news/security-analysis-of-the-solo-firmware-by-doyensec, February 2020.

[STM17] STMicroelectronics. *RM0394: STM32L43xxx STM32L44xxx STM32L45xxx STM32L46xx advanced ARM ®-based 32-bit MCUs*, April 2017. Rev. 3.

[Sue12] Ruth Suehle. Was Your LinkedIn Password Leaked? https://www.wired.com/2012/06/linkedin-data-breach/, June 2012.

[Suh20] Jan Suhr. Breaking the STM32F1 Read-Out Protection. https://support.nitrokey.com/t/breaking-the-stm32f1-read-out-protection/2186, February 2020.

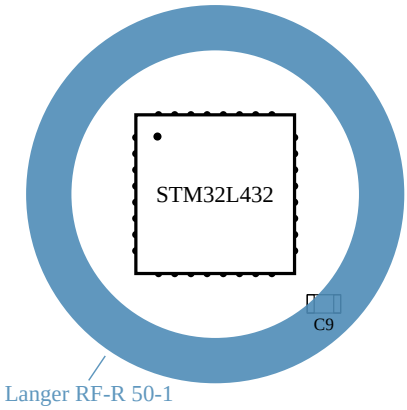
[Yub17] Yubico. Security advisory YSA-2017-01 – Infineon weak RSA key generation. https://www.yubico.com/support/security-advisories/ysa-2017-01/, 2017.

[Yub19] Yubico. Security advisory YSA-2019-02 – reduced initial randomness on FIPS keys. https://www.yubico.com/support/security-advisories/ysa-2019-02/, 2019.

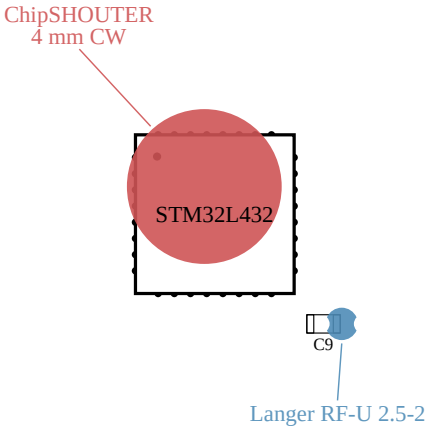
[Zerc4] U2F Zero. https://github.com/conorpp/u2f-zero, Commit: fd282c4.

A Side-Channel and Fault Injection Positions

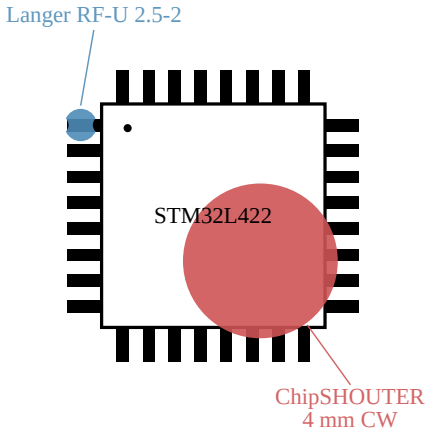
The probe position in EM side-channel and fault injection setups determines the results to a large extent. In order to make our work comprehensible and reproducible for other researchers, Figure 10 shows the respective probe positions of our setups.



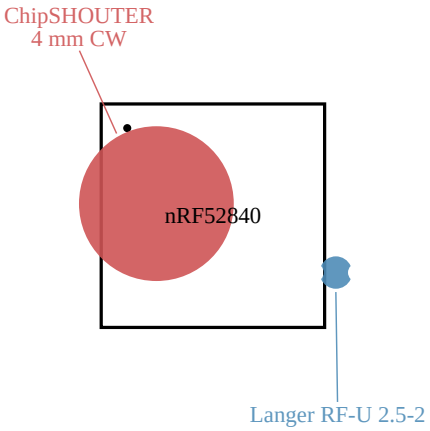
(a) Key extraction using SCA on the Solo security token.



(b) Key extraction using EMFI on the Solo security token.



(c) Mass erase bypass using EMFI on the STM32L422 microcontroller.



(d) Key extraction using EMFI on the nRF52840 microcontroller.

Figure 10: Schematic top view of the examined devices. Blue highlights the position of the EM probe for the side-channel analysis. Red highlights the position of the EM injection coil for the fault injection attacks.

B Flash Memory Extraction of Nitrokey Pro 2

The microcontroller integrated in the Nitrokey Pro 2 exhibits a vulnerability [SO20] in its flash memory read-out protection. This vulnerability enables an adversary to extract the OTP entries including the respective secrets from the flash memory. Due to the nature of this vulnerability, it is not possible to read out every entry completely. In Table 3, the number of bytes for each OTP entry that cannot be extracted are listed.

Table 3: Number of bytes for each OTP entry that cannot be extracted from the flash memory of the Nitrokey Pro 2. The fields represent the `struct` members of each entry as used by the firmware. Entries that can be fully extracted are omitted for clearness.

OTP field	HOTP		TOTP								
	2	3	1	2	4	5	7	9	10	12	13
`type`	0	0	0	0	0	1	0	0	1	0	0
`name`	0	8	0	15	0	0	9	0	5	1	0
`secret`	8	4	4	9	0	24	15	0	24	0	19
`slot_number`	0	0	0	0	0	0	0	0	1	0	0
`config`	0	0	1	0	0	0	0	0	0	0	1
`token_id`	9	0	3	0	0	0	0	0	0	0	9
`interval_or_counter`	7	0	0	0	7	0	0	1	0	0	3

C Coordinated Disclosure

As part of a coordinated disclosure process, we informed the security teams of all affected products about our findings. Technical and detailed information were provided more than 90 days prior to the publication of this paper. In Table 4, the assigned CVE numbers for each identified vulnerability together with the affected product(s) and the corresponding manufacturer are listed.

Table 4: Assigned CVE numbers for each vulnerability identified in this work.

Manufacturer	Product	CVE numbers
STMicroelectronics	STM32L4	CVE-2020-27212, CVE-2021-29414
SoloKeys	Solo	CVE-2020-27208
	Somu	
Nitrokey	Nitrokey FIDO U2F	CVE-2020-12061
	Nitrokey FIDO2	CVE-2020-27208
Nordic Semiconductor	nRF52840	CVE-2020-27211, CVE-2021-29415[a]
-	micro-ecc	CVE-2020-27209

[a] As there was no CVE assigned, we include the previous research of [Lim20].