Practical Fault Injection on Deterministic Signatures: The Case of EdDSA

Niels Samwel(✉) and Lejla Batina(✉)

Digital Security Group, Radboud University, Nijmegen, The Netherlands
{n.samwel,lejla}@cs.ru.nl

Abstract. After recent vulnerabilities of implementations of deterministic signatures e.g. EdDSA have been revealed, it became evident that a secure deployment of those will require additional countermeasures. Nevertheless, this is not a simple task, as we show in this work. We demonstrate the easiness of fault attacks on EdDSA as implemented in the lightweight cryptographic library WolfSSL on a 32-bit micro-controller. We achieve a success rates of almost 100% by voltage glitching and electromagnetic fault injection. Even after adding certain checks as a countermeasure, the implementation remains vulnerable to fault injection. As only a single successful fault is needed to recover the key, this kind of implementation is an easy target for the attackers.

Keywords: ECC · EdDSA · Differential fault attack

1 Introduction

In our daily lives the use of small embedded devices have become prevalent due to their numerous deployments in transportation, secure payments and e-health systems, as wearables etc. The accessibility of those devices makes them a perfect target for a side-channel adversary who is able to collect and process leakage signals leading to the secret/private data recovery. On top of this, the protection against this kind of adversary is complicated due to the sparseness of resources such as area, memory, power/energy budgets etc.

Typical services for the IoT and other embedded devices include authentication, which sometimes also needs to be performed off-line. One way to enable strong authentication is to use digital signatures, where Elliptic Curve Cryptography (ECC) is still leading the field for lightweight Public-key cryptosystems (PKC). One of the best known signature algorithms is due to Schnorr [25], which was introduced for discrete logarithm cryptosystems. Other signature schemes have been also proposed such as the Digital Signature Algorithm (DSA) [18]. Later this scheme was extended to a scheme called ECDSA [15] that is using elliptic curves. DSA-like signature schemes require a fresh randomly generated ephemeral key for each signature.

The ephemeral key used in DSA-like schemes has to be truly random. Some recent studies showed how real-world system do not always follow this recommendation [13]. The requirement turns out to be a complex issue, especially for resource constrained devices that may not have a true random number generator. Actually, if only a few bits of the ephemeral key are known the private key can be recovered using a specific lattice-based cryptanalysis [14]. To downplay the importance of the true randomness of the ephemeral key an alternative to ECDSA so-called EdDSA was introduced [8]. The selling point of EdDSA is that the ephemeral key is generated deterministically and the requirement for cryptographically secure randomness becomes obsolete. However, a recent side-channel attack on EdDSA has shown that the deterministic feature is not optimal in practice with the adversary that has an access to the device and is able measure side-channel signals [24]. Namely, the promoted feature to make EdDSA deterministic complicates its secure implementation as it makes it a clinical use case for a first order DPA attack using power or EM leakages.

In addition, fault attacks on deterministic signature schemes also appeared recently by Ambrose et al. [2] and Romailler and Pelissier [23]. The former outlines theoretically several scenarios for different fault attacks on deterministic signatures in contrast to the latter which describes a very special practical attack on EdDSA that is feasible on an 8-bit platform only.

The work we present in this paper is a generic fault injection attack that can be applied to a range of platforms and it is using different sources for fault injection. We demonstrate the pervasiveness of it on a 32-bit micro-controller targeting EdDSA implementation within the lightweight cryptographic library WolfSSL. For our attack a single fault during the scalar multiplication algorithm is required for the full key recovery. We give all the details of the setup where this semi-invasive attack is done by applying minor changes to the supply voltage or using electro-magnetic EM signals as the "glitching" sources.

The rest of this paper is organized as follows. First we list related previous work and specify our contributions. In Sect. 2, we provide background information required for the remainder of the paper. Section 3 presents an overview of the attack and more detailed methodology on voltage fault injection and electromagnetic fault injection. Section 4 shows the results of our attack. In Sect. 5, we discuss several countermeasures against this attack and fault attacks in general. Section 6 concludes this paper.

1.1 Related Work

In 1997, the first differential fault attack on public key system RSA-CRT was introduced by Boneh et al. [11]. The authors presented a theoretical concept together with a possible countermeasure. Later Aumüller et al. [3] show the feasibility of the attack by applying it in practice and presenting another countermeasure.

Considering other PKC the first differential fault attack on an elliptic curve cryptosystem was presented by Biehl et al. [10] in 2000. In the scenario they propose the resulting point is not on the original curve anymore. Hence, as a consequence they validate the point as a countermeasure.

Barenghi and Pelosi [5] describe several potential fault attacks on EC-based signature schemes theoretically. In one of the attacks, a fault is introduced during the computation of the hash function. This value is not public and must be recovered by brute forcing over all possible values. The authors implemented the key recovery part and presented their results for this specific scenario.

Recently, a work by Ambrose et al. [2] outlined several differential fault attacks on deterministic signature schemes. However, the authors present no practical results.

The first differential fault attack on Ed25519 was published by Romailler and Pelissier [23]. The authors used the Arduino nano, an 8-bit micro-controller as their target platform where a signing operation takes over 5 s. They introduced a fault in the output of a hash function which is not public so the requirement for the attack is to brute force this value. This issue is complicated with modern platforms using 32- or 64-bit architectures. Therefore, the attack is not so practical for other than 8-bit architectures. In our attack, we introduce a fault during the scalar multiplication which makes it platform-independent.

With the introduction of the Rowhammer attack [17] several papers have been published on injecting faults using software manipulations. Poddebniak et al. [21] used the idea to attack deterministic signature schemes. In their work, they explain how to apply the Rowhammer attack and how to prevent it by presenting several countermeasures. This attack is very different than our attack because the impact of Rowhammer is that an invalid signature is generated, while our attack recovers the relevant part of the key in order to forge a signature.

1.2 Contributions

Here we summarize the main contributions of this paper:

- In this paper we present a conceptually novel and generic differential fault attack on the deterministic signature scheme Ed25519. We inject the fault in the scalar multiplication operation that is unrelated to the hash computation, such that the attacker does not need to brute force the intermediate result.
- The attack is demonstrated on a real-world implementation of Ed25519 from the lightweight cryptographic library WolfSSL on a 32-bit micro-controller. This kind of implementation particularly targets low-cost and/or resource-constrained environments as in the IoT use cases and similar.
- We show that our attack can be effectively executed using voltage glitching and electromagnetic fault injection.
- We also establish the fact on the necessity of suitable countermeasures as we show that even the common point validity check countermeasure cannot counteract the attack.

2 Background

2.1 EdDSA

EdDSA is a known digital signature scheme constructed over so-called Edwards curves [8]. An instance of EdDSA using Edwards Curve25519 in particular (called

Ed25519) is used in Signal protocol, Tor, SSL, etc. There is also an ongoing effort to standardize the scheme, known as RFC 8032. The signature scheme is a variant of the Schnorr signature algorithm [25] that makes use of Twisted Edwards Curves. Compared to ECDSA, EdDSA does not need new randomness for each signature as the ephemeral key is computed deterministically using the message and the auxiliary key that is part of the private key. The security depends on the secrecy of the auxiliary key and the private scalar. This does not create an additional requirement as we need to keep a private key secret anyway. The security of ECDSA depends heavily of a good quality randomness of the ephemeral key, which has to be truly random for each signature. This feature was put forward in promotion of EdDSA as being more side-channel resistant than ECDSA [8].

In Ed25519, a twisted Edwards curve birationally equivalent to Curve25519 [7] is used. Ed25519 sets several domain parameters of EdDSA such as:

- Finite field F_q, where $q = 2^{255} - 19$
- Elliptic curve $E(F_q)$, Curve25519 [7]
- Base point B
- Order of the point B, l
- Hash function H, SHA-512 [22]
- Key length $b = 256$

For more details on other parameters of Curve25519 and the corresponding curve equations we refer to Bernstein [7].

Table 1. Notations EdDSA

Name	Symbol
Private key	k
Private scalar	a (first part of $H(k)$)
Auxiliary key	b (last part of $H(k)$)
Ephemeral scalar of private key	r

To sign a message, the signer has a private key k and message M. Algorithm 1 shows the steps to generate an EdDSA signature.

The first four steps belong to the key setup and are only applied the first time a private key is used. Notation $(x, \ldots, y)$ denotes concatenation of the elements. We call a the private scalar and $b = (h_0, h_1, \ldots, h_{2b-1})$ the auxiliary key (see Table 1). In Step 5 the ephemeral key is deterministically generated.

To verify a signature (R, S) on a message M with public key A a verifier follows the procedure described in Algorithm 2.

Algorithm 1. EdDSA signature generation

```
   Key setup.
1: Hash $k$ such that $H(k) = (h_0, h_1, \ldots, h_{2b-1})$.
2: $a = (h_0, \ldots, h_{b-1})$, interpret as integer in little-endian notation.
3: $b = (h_b, \ldots, h_{2b-1})$.
4: Compute public key: $A = aB$.
   Signature generation.
5: Compute ephemeral key: $r = H(b, M)$.
6: Compute ephemeral public key: $R = rB$.
7: Compute $h = H(R, A, M)$ and convert to integer.
8: Compute: $S = (r + ha) \mod l$.
9: Signature pair: $(R, S)$.
```

Algorithm 2. EdDSA signature verification

```
1: Compute $h = H(R, A, M)$ and convert to integer.
2: Check if group equation $8SB = 8R + 8hA$ in $E$ holds.
3: If group the equation holds, the signature is correct.
```

2.2 Fault Attacks

Fault attacks are active attacks and aim at exploiting the leakage of sensitive information due to some irregular conditions i.e. faulty computation. This is distinctive to side-channel attacks that observe signals while the device under attack is working "normally". With fault attacks, an attacker attempts to alter environmental conditions so the device changes it behavior. One way to accomplish this is by "glitching" the device i.e. forcing the changes in the values of relevant physical parameters outside the prescribed intervals. There are several approaches to accomplish this as follows:

– **Clock fault injection** [1]**.** In this case a glitch is caused by altering the clock signal. This is typically done with devices that allow the use of an external clock.
– **Voltage fault injection** [3]**.** The attacker can induce this kind of glitch by adding a short positive or negative spike in the power line.
– **Electromagnetic fault injection** [27]**.** A glitch is caused by emitting a short electromagnetic (EM) pulse towards the device resulting in similar effects as voltage glitching.
– **Optical fault injection** [26]**.** Optical fault injection is more invasive as the chip typically has to be decapsulated and it often causes permanent damage to a device.

In this paper we focus on glitches caused by voltage and electromagnetic fault injection. If a glitch has an effect that alters the behavior of the device such that it produces a fault, we call this "a successful fault".

Fault Model. A fault model describes the kind and the extent of faults an attacker is able to induce while the device is operating. In this paper our target platform is a micro-controller with a 32-bit architecture so we assume a fault model where a glitch can create an error such that the value of a 32-bit word is modified. This alteration can happen at different stages, for instance when the value is processed by the CPU or when the value is on the memory bus. Typically, a glitch could also alter instructions that affect other sensitive values (e.g. loop counters etc.) or other behavior of the algorithm but our attacks do not rely on this particular assumption.

Differential Fault Analysis. Differential fault analysis (DFA) is a special attack based on faults produced during computation. Typically, the attacker uses the difference between the correct output and one (or more) faulty outputs to recover secret data. This can lead to the total key recovery like in the case of Bellcore attacks on the RSA cryptosystem [11] or merely to forging a signature.

3 Methods

3.1 General Attack Principle

In Ed25519, if an attacker is able to cause a glitch in the computation of the ephemeral public key $R = rB$, or in the computation of the hash $h = H(R, A, M)$ where the same message is signed resulting in R' or h', he can recover the private scalar. Independent of which of the two values is faulty, the hash computation is always faulty as it has R or R' as an input. For a successful attack, we need a correct signature and only a single faulty signature to recover the private scalar a. With private scalar a, a valid signature on any message can be computed as the value of r is arbitrary. From the correct and a faulty signature, the private scalar a can be recovered as follows. The attacker obtains a correct signature (R, S) and a faulty signature (R', S') with the following equations:

$$S = r + ha,$$
$$S' = r + h'a.$$

If we rewrite this, we obtain the following,

$$S - ha = S' - h'a.$$

And we can extract private scalar a

$$a = \frac{S - S'}{h - h'} \tag{1}$$

The output of the hash function h is not public so when a fault is injected in the computation of the hash, an attacker must know or be able to compute hash h'. It can be brute forced as in [23] where the authors use an 8-bit architecture, but their attack does not scale so when a more realistic target is used like a 32-bit or 64-bit architecture, this becomes impractical. Since the ephemeral public key R is part of the signature (hence known), we aim at causing a glitch in the computation of the scalar multiplication $R = rB$. We do not target any particular single bit (or a group of bits), but a fault in any intermediate value of the scalar multiplication is sufficient.

For each execution of the signing algorithm there are three possible outcomes.

- Normal
- Inconclusive
- Successful

A normal outcome denotes the case when no fault occurred and the output is as expected. A successful outcome stands for an induced fault that resulted in the correct key by applying Eq. (1). An inconclusive outcome has several possibilities: (i) a fault was induced and a faulty output was produced but the key could not be recovered, (ii) a fault was induced but no output was produced, and (iii) a fault was induced but no output was produced and the device stopped working. At this point the device had to be power cycled to continue the experiment.

3.2 Voltage Fault Injection

We start with finding the lowest VCC (VCC_L) for which the target still behaves "normally". Next, we under-power the device continuously (VCC_F) so that it still works but faults are introduced, see Table 2 for the settings. Our goal is to maximize the success rate.

Figure 1 shows a schematic overview of the setup. The PC handles communication with the target, collects traces from the oscilloscope and controls the VC Glitcher. The target is powered by the VC Glitcher so it is able to inject glitches in the power line. The glitch amplifier amplifies the current and with the current probe we are able to measure the power consumption and see the effects of the glitch. The oscilloscope and the current probe are only used to collect an overview trace and determine an offset to induce the glitch. Once a suitable offset is located, the current probe and oscilloscope are disconnected and removed from the setup.

Once we identify this offset we try to improve the success rate and we continuously under-power the device by actively inducing faults. To do this we introduce a glitch after a trigger event occurs. We set a trigger at the start of the scalar multiplication in the signature generation. To maximize the success rate there are several parameters to optimize, such as:

- Glitch Voltage (GV),
- Glitch Length (GL),
- Glitch Offset (GO),
- Glitch Repetition (GR).

Table 2. Settings for voltage fault injection setup.

Name	Setting
VCC_L	2.3 V
VCC_F	2.201 V
Glitch Voltage	−0.16 V
Glitch Length	3070 ns
Glitch Offset	1444010 ns
Glitch Repetition	1

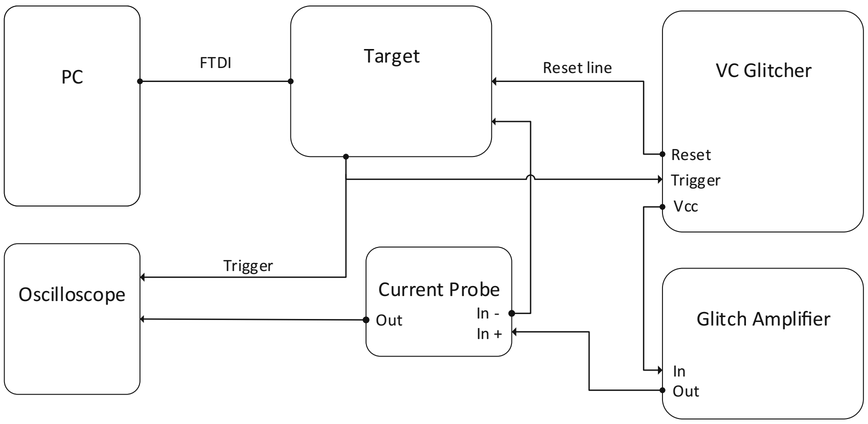


Fig. 1. Voltage fault injection setup

In the experiments we introduce single glitch so we fix the glitch repetition to 1. To optimize those parameters, we did not apply any sophisticated algorithm such as e.g. [12,20], but we applied random search instead. With the first results we manually narrowed down the search ranges to find optimal parameters.

3.3 Electromagnetic Fault Injection

Electromagnetic fault injection (EMFI) is an active attack where the attacker emits a short EM pulse as a glitch from a close distance. If the glitch is strong enough, it can cause a fault. The EM pulse is emitted using a small coil. Different coils could have different effects on the size and the polarity of the EM pulse, but this point is not relevant for our work.

As with voltage fault injection, there exist several parameters to be optimized for EMFI. Those are also different parameters and the most distinctive ones are the x and y coordinates corresponding to a location on the chip where the coil that emits the EM pulse is positioned. Figure 2 shows an overview of the setup. We use the XY-table to precisely position the EMFI probe on the device. With the XY-table we are able to automate a systematic scan of the chip's surface to

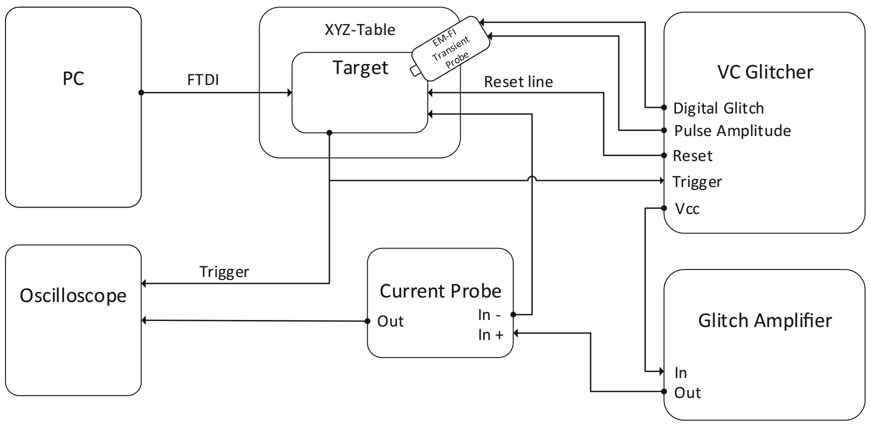


Fig. 2. EMFI setup

find a location with the highest success rate of the attack. Below we list all the parameters we need to optimize.

– x-Coordinate
– y-Coordinate
– Glitch Power (GP)
– Glitch Offset (GO)
– Glitch Repetition (GR)
– Glitch Length (GL)

Again we introduce a single glitch so we fix glitch repetition to 1. The parameter glitch length is fixed to 40 ns due to the EMFI hardware that we used in the attack.

4 Experimental Setup and Results

4.1 Setup

The setups for voltage and EM fault injection are very similar. Our target is a development board containing a Cortex-M4F, more specifically the STM32F407IG. For our experiments we did not have to decapsulate the chip. A signing operation of Ed25519 from WolfSSL takes roughly 30 ms on this platform. Electronic devices have capacitors to keep the power at a stable level so internal or external fluctuations do not influence the behavior of the device. Since we actually want to cause some fluctuations in the power line to alter the behavior with voltage FI, we removed most capacitors on the board. With EMFI we externally cause the fluctuations in the power plane with short EM pulses so the attack also works without removing the capacitors.

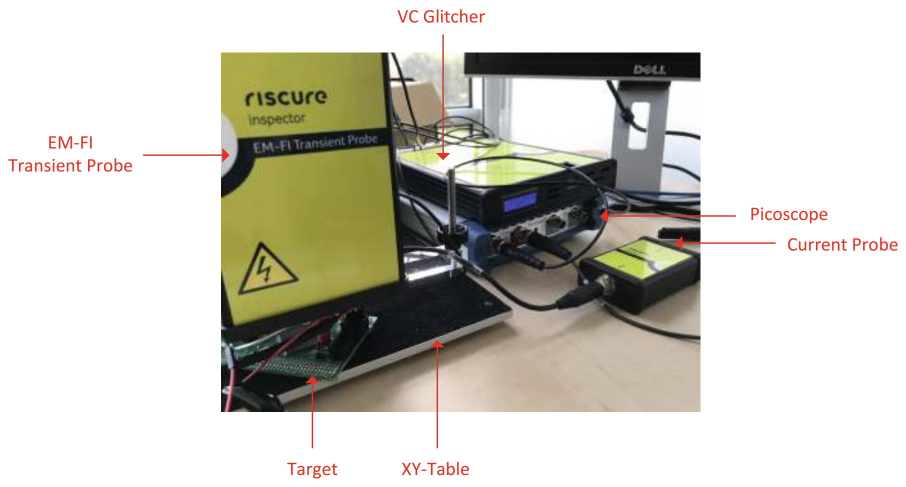


Fig. 3. This figure shows the experimental setup. In the top left corner we see the EM-FI transient probe and below that the target board which is fixed to the XY-table. In the center with the blue screen, we see the VC Glitcher under witch is the oscilloscope and the small block on the right is the current probe. (Color figure online)

We use the VC Glitcher[1] to power the board and to cause fluctuations in the voltage. We also need the Glitch Amplifier as the VC Glitcher does not provide enough current to power the board.

To generate the EM pulses we use an EMFI Transient Probe that is connected and controller by the VC Glitcher. An *xy*-table is used to move the EMFI Transient Probe with high accuracy.

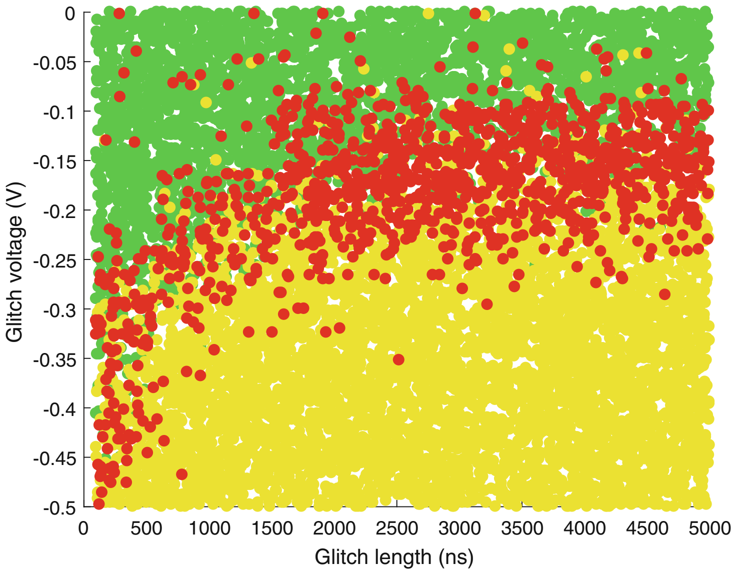
The oscilloscope used to visualize the effect of the voltage fluctuations or the EM pulses is a Picoscope5203. A current probe measuring those changes is connected in series with the power line and it is also a part of the setup. Figure 3 shows a picture of our experimental setup.

We attack the software implementation of Ed25519 in WolfSSL version 3.11.0. Similar implementations can be found in other cryptographic libraries implementing Ed25519. We added a trigger to the code right before the start of the scalar multiplication. We could also have inserted the trigger before the signature generations starts as the code runs in constant time, so it would merely imply the increase in the offset. The attack is also possible without adding a trigger in the code when using hardware that can generate a trigger based on a pattern in the signal [6].

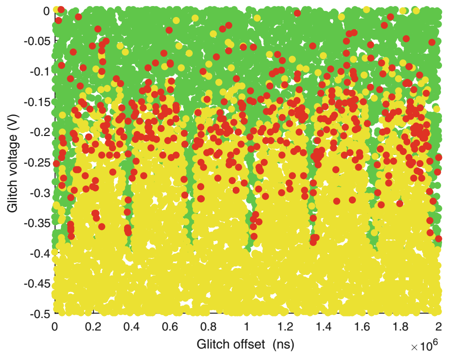
4.2 Voltage Fault Injection Results

The first step of the experiment was to continuously under-power the device. Without introducing glitches we were able to achieve a success rate of 44%. When we actively tried to induce glitches using the described parameters, we

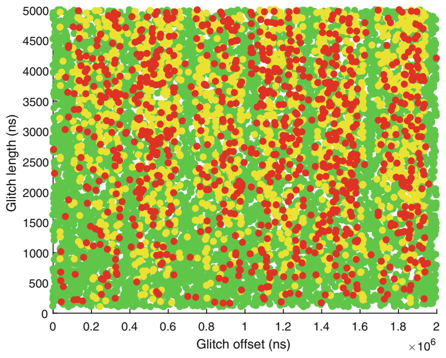
[1] https://www.riscure.com/security-tools/hardware/.



(a)



(b)



(c)

Fig. 4. Voltage fault injection results, Normal (green), Inconclusive (yellow), Successful (red). (Color figure online)

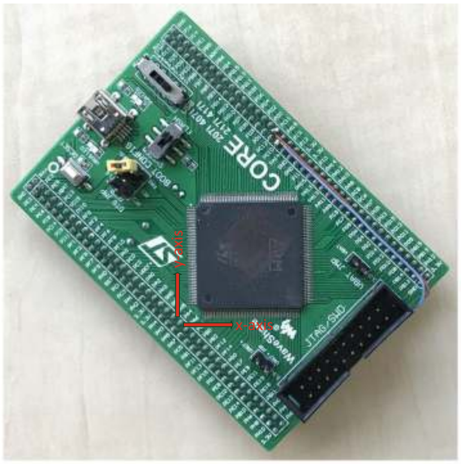
were able to increase the success rate to 69.95% using an optimal set of parameter values we found. We computed the success rates using 10 000 measurements with those optimal parameters.

To visualize the effects of the glitch parameters, we set the parameters to a constant value except for two. For those two parameters we selected random value within a certain range. Figure 4 shows the result of the experiment. In Fig. 4a, we vary the glitch length and the glitch voltage parameters. It shows a typical curve of successful glitches as in [12, 20]. A selected set of parameters above the curve means the glitch is not strong enough and the device continues like nothing happened and a selection of parameters below the curve results in a glitch that is too strong and the device stops responding. In Fig. 4b and c we vary offset with the glitch voltage and glitch length and we see a clear pattern that corresponds to an iteration in the scalar multiplication. Each figure contains results of 10 000 measurements.

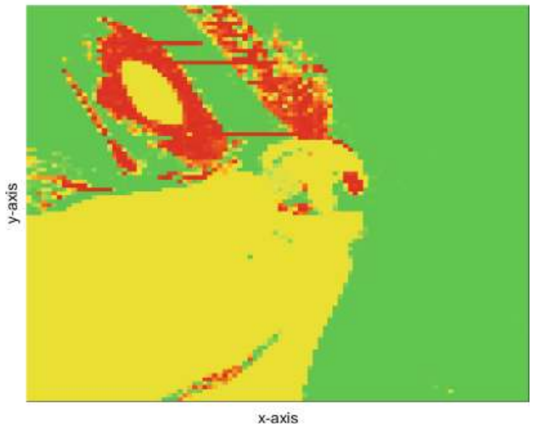
The results show that inducing exploitable faults is not complicated as even providing a lower voltage results in a reasonably high success rate.

4.3 Electromagnetic Fault Injection Results

In this experiment we start by scanning the surface of the chip to determine good positions for a successful fault injection. Figure 5a shows the surface of the chip.



(a) Target board



(b) Heat map with EMFI results

Fig. 5. This figure shows a picture of the board and it shows which locations are most effective to inject a glitch.

We divide the x and y-axis up in 100 parts each, resulting in 10 000 positions to scan. We inject a glitch 20 times on each position where the remaining parameters are randomized similar as in the previous section. By manually optimizing the parameters, we were able to achieve a success rate of 99.31%. We did another surface scan with these fixed parameters, the result is shown in Fig. 5b. The figure shows a heatmap, where the color denotes the result of a fault. A color that is a mix between other colors is corresponding to the situations when the resulting faults were mixed.

To scan the surface we performed a total amount of 200 000 measurements. With the best parameters choice of paremetrs, we used 10 000 measurements to compute the success rate.

Since we induce a fault in the scalar multiplication, we expected most resulting points would not be on the curve anymore. In WolfSSL there is no check like this implemented, so we added it ourselves to let the signing operation fail in case of a fault as a countermeasure. We scanned the surface again with optimal parameters and as expected not a single fault was successful.

To have a faulty scalar multiplication where the resulting point still is on the curve, we can modify the scalar itself. However, this is only possible while it is used within the scalar multiplication as the original scalar r must be used to compute S that is leading to the key recovery. In the implementation of WolfSSL (similar in other implementations) the scalar is copied and some computations are done to alter its representation. This takes roughly 36μs and gives us plenty of time to emit an EM pulse while the original scalar remains unaltered. Optimizing the parameters resulted in a success rate of 70.15%. Although the success rate with the check on the validity of the resulting point is lower, it is still high enough for the attack to remain practical.

5 Countermeasures

There are several approaches to count fault injection attacks, both in hardware [9,16] and in software [3,4]. Here we discuss some software countermeasures.

A countermeasure is to add redundancy in the implementation. For instance, in a common countermeasure, the implementation could execute the scalar multiplication again and at the end compare both results. If they are not identical, then the fault occurred and the signature should not be released. The signature could also be verified at the end, if the signature is invalid, do not return the signature.

However, there is a problem with adding redundancy as a countermeasure as this introduces a check in the code that an attacker also could try to skip by injecting a glitch. It also penalizes the performance. Although this adds a significant amount of difficulty to the attack, an attacker only has to be successful once to be able to recover the secret key.

Another solution is to add randomness to the scheme. In [23] the authors propose a countermeasure called "fault infective computations" where 32 random bytes are used together with different implementations of the hash function for each time the hash function is used in the scheme. A second implementation of the hash function adds to the code size and may not be preferred due to resource constraints.

In this attack we exploit that ephemeral scalar r is equal in both signatures. Introducing some randomness in the generation of r counters our attack. New standards where randomness is introduced in the generation of r are proposed like XEdDSA and VXEdDSA [19]. In these schemes 64 random bytes are added, the VXEdDSA scheme also requires several additional scalar multiplications. In [24], the authors propose a cheap countermeasure that requires only 16 random bytes that are used in the generation of r. The randomness does not have to be perfect as with ECDSA as long as the bytes remain unknown to an attacker. On top of that, the countermeasure also protects the key against differential power analysis [24]. Signatures generated using this countermeasure are still verifiable and conformed with the standard.

6 Conclusion

With this paper we improve and generalize previous attacks on Ed25519, using a realistic target platform and a real-world implementation. We show that our attack is possible using voltage FI and EM FI with very high success rates. While we are able to achieve high success rates, close to 100%, we would like to note that an attacker only needs a single successful fault to recover the key. With adding redundancy to the implementation, it would remain in agreement with the standard but still vulnerable to fault injection. To counter fault injection, the standard should be modified to add some randomness in the generation of the ephemeral scalar.

Although we attack the implementation of WolfSSL, the attack is extendable to other implementations of Ed25519 as the issues are with the scheme being implemented straightforwardly and not a particular implementation itself.

Acknowledgments. This work was supported in part by the Technology Foundation STW (Projects 13499 TYPHOON and 12624 SIDES) and The Netherlands Organization for Scientific Research NWO (project ProFIL 628.001.007) and by a project funded by DarkMatter LLC.

References

1. Agoyan, M., Dutertre, J.-M., Naccache, D., Robisson, B., Tria, A.: When clocks fail: on critical paths and clock faults. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 182–193. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12510-2_13
2. Ambrose, C., Bos, J.W., Fay, B., Joye, M., Lochter, M., Murray, B.: Differential attacks on deterministic signatures. Cryptology ePrint Archive, Report 2017/975 (2017). https://eprint.iacr.org/2017/975.pdf

3. Aumüller, C., Bier, P., Fischer, W., Hofreiter, P., Seifert, J.-P.: Fault attacks on RSA with CRT: concrete results and practical countermeasures. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 260–275. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_20
4. Barenghi, A., Breveglieri, L., Koren, I., Pelosi, G., Regazzoni, F.: Countermeasures against fault attacks on software implemented AES. In: Proceedings of the 5th Workshop on Embedded Systems Security - WESS 2010. ACM Press (2010)
5. Barenghi, A., Pelosi, G.: A note on fault attacks against deterministic signature schemes (short paper). In: Ogawa, K., Yoshioka, K. (eds.) IWSEC 2016. LNCS, vol. 9836, pp. 182–192. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44524-3_11
6. Beckers, A., Balasch, J., Gierlichs, B., Verbauwhede, I.: Design and implementation of a waveform-matching based triggering system. In: Standaert, F.-X., Oswald, E. (eds.) COSADE 2016. LNCS, vol. 9689, pp. 184–198. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43283-0_11
7. Bernstein, D.J.: Curve25519: new Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer, Heidelberg (2006). https://doi.org/10.1007/11745853_14
8. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. J. Crypt. Eng. **2**(2), 77–89 (2012)
9. Bertoni, G., Breveglieri, L., Koren, I., Maistri, P., Piuri, V.: Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. IEEE Trans. Comput. **52**(4), 492–505 (2003)
10. Biehl, I., Meyer, B., Müller, V.: Differential fault attacks on elliptic curve cryptosystems. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 131–146. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_8
11. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_4
12. Carpi, R.B., Picek, S., Batina, L., Menarini, F., Jakobovic, D., Golub, M.: Glitch it if you can: parameter search strategies for successful fault injection. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 236–252. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_16
13. Checkoway, S., Maskiewicz, J., Garman, C., Fried, J., Cohney, S., Green, M., Heninger, N., Weinmann, R.P., Rescorla, E., Shacham, H.: A systematic analysis of the Juniper Dual EC incident. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016, pp. 468–479 (2016). http://doi.acm.org/10.1145/2976749.2978395
14. Howgrave-Graham, N.A., Smart, N.P.: Lattice attacks on digital signature schemes. Des. Codes Crypt. **23**(3), 283–290 (2001). https://doi.org/10.1023/A:1011214926272
15. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). Int. J. Inf. Secur. **1**(1), 36–63 (2001)
16. Karpovsky, M., Kulikowski, K., Taubin, A.: Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard. In: 2004 International Conference on Dependable Systems and Networks. IEEE (2004)
17. Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J.H., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping bits in memory without accessing them. ACM SIGARCH Comput. Archit. News **42**(3), 361–372 (2014)

18. Kravitz, D.: Digital signature algorithm. US Patent 5,231,668, 27 July 1993. https://www.google.com/patents/US5231668
19. Perrin, T.: The XEdDSA and VXEdDSA Signature Schemes (2017). https://signal.org/docs/specifications/xeddsa/xeddsa.pdf. Accessed 11 Sept 2017
20. Picek, S., Batina, L., Jakobovic, D., Carpi, R.B.: Evolving genetic algorithms for fault injection attacks. In: 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE, May 2014
21. Poddebniak, D., Somorovsky, J., Schinzel, S., Lochter, M., Rösler, P.: Attacking deterministic signature schemes using fault attacks. Cryptology ePrint Archive, Report 2017/1014 (2017). http://eprint.iacr.org/2017/1014
22. FIPS PUB 180-4: Secure Hash Standard (SHS). Technical report, NIST, July 2015
23. Romailler, Y., Pelissier, S.: Practical fault attack against the Ed25519 and EdDSA signature schemes. In: 2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). IEEE, September 2017
24. Samwel, N., Batina, L., Bertoni, G., Daemen, J., Susella, R.: Breaking Ed25519 in WolfSSL. Cryptology ePrint Archive, Report 2017/985 (2017). http://eprint.iacr.org/2017/985
25. Schnorr, C.P.: Efficient signature generation by smart cards. J. Crypt. **4**(3), 161–174 (1991). https://doi.org/10.1007/BF00196725
26. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 2–12. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_2
27. Velegalati, R., Van Spyk, R., van Woudenberg, J.: Electro magnetic fault injection in practice. In: International Cryptographic Module Conference (ICMC) (2013)