Efficient and Secure Elliptic Curve Cryptography for 8-bit AVR Microcontrollers

Erick Nascimento, Julio López, and Ricardo Dahab

Institute of Computing, University of Campinas, Campinas, Brazil
ra032483@students.ic.unicamp.br,
{jlopez,rdahab}@ic.unicamp.br

Abstract. The AVR family of 8-bit microcontrollers is widely used in several applications demanding secure communications and protection against physical attacks, such as side-channel analysis. In this context, processing, storage and energy demands of cryptographic software must be low, requirements which are met by ECC. At the 128-bit security level, two recently proposed curves are an attractive option for 8-bit microcontrollers: Curve25519 for Diffie-Hellman key exchange, and Ed25519 for signature. Simple power analysis is a significant threat to AVR applications, but efficient and side-channel tested implementations of SPA countermeasures for ECC protocols have not yet been dealt with in this platform, in the literature. This paper describes an efficient implementation of ECDH-Curve25519 and EdDSA-Ed25519-SHA512 for the ATmega328P platform. Our implementation provides protection against timing attacks, SPA and template SPA. The resistance against SPA is evaluated through the test vector leakage assessment (TVLA) methodology based on Welch's t-test, using the Chipwhisperer platform.

Keywords: Public-key cryptography, elliptic curves, ECDH, EdDSA, embedded system, AVR, side-channel attack, timing analysis, simple power analysis, SPA, template SPA, countermeasure.

1 Introduction

Elliptic Curve Cryptography (ECC) is a class of public-key cryptosystems proposed by Koblitz [32] and Miller [40], which provides significant efficiency advantages for microcontrollers, due to small key sizes which may improve speed, memory and power. For example, some industry standards require 2048-bit keys for RSA, whereas the equivalent security for ECC demands 224-bit keys. In fact, ECC-based protocols are used in many embedded applications, such as payment, pay-TV, wireless sensors, medical and identification systems.

Passive side-channel attacks (SCA) are a class of implementation attacks exploiting physical leakages of a device during the execution of a cryptographic operation, such as: timing [33], power consumption [34] and electromagnetic radiation [47,21]. They present a realistic threat to cryptographic applications, and have demonstrated to be very effective against smart cards without proper

© Springer International Publishing Switzerland 2015
R.S. Chakraborty et al. (Eds.): SPACE 2015, LNCS 9354, pp. 289–309, 2015.
DOI: 10.1007/978-3-319-24126-5_17

countermeasures [37]. Evaluation of SCA resistance is mandatory in some current and upcoming standards: Common Criteria [17], FIPS 140-3 [43] and others [31]. SCA attacks can be classified in two categories: *Simple Side-Channel Analysis* (SSCA) [33], in which measurements (traces) obtained for a single or few runs of a private key operation (e.g., signing or decryption) are acquired, and the differences in the measured physical quantity depending on the value of the secret key are analyzed; and *Differential Side-channel Analysis* (DSCA) [34], which is based on statistical analysis to retrieve information about the private key based on a large number of traces. SSCA is considered the main side-channel threat against implementations of public key cryptographic algorithms.

Current cryptographic standards require a work factor around 128 bits [2,11] [44,45]. Curve25519 [5] and Ed25519 [6] are two curves at the 128 bit security level that have achieved promising industry adoption. Curve25519 is a curve in the Montgomery model over the 255-bit prime field $\mathbb{F}_p$, for $p = 2^{255} - 19$, suitable for ECDH. Ed25519 is a curve in the twisted Edwards model also defined over $\mathbb{F}_p$, but designed for the EdDSA (Edwards DSA) signature scheme [6].

Related Work. The closest related works that can be directly compared to ours are the port of the NaCl library to AVR by Hutter and Schwabe [27], which includes constant time implementations of both EdDSA-Ed25519 (23 216 241 cycles for signing and 32 634 713 cycles for verification) and ECDH-Curve25519 (22 791 579 cycles for computing a shared secret key using Montgomery ladder) protocols, and the faster version [19] of ECDH-Curve25519 (13 900 397 cycles using Montgomery ladder). Other implementations of ECC for twisted Edwards or Montgomery curves for 8-bit AVRs in the literature cannot be directly compared to ours, because they target different curves. Chu et al [14] implemented ECC for twisted Edwards curves over 160 and 192 bits Optimal Prime Fields (OPFs), but do not implement any countermeasures against SCA. Liu et al [35] described an implementation of ECC for Montgomery curves over OPFs with field sizes ranging from 160 to 256 bits[1]. Liu et al [36] described constant time, variable and fixed-base scalar multiplications for twisted Edwards and Montgomery curves over OPFs, the latter uses a highly regular comb algorithm.

Our Contributions. We describe implementations of fixed and variable-base elliptic curve scalar multiplication algorithms for Ed25519, and of EdDSA-Ed25519-SHA512 signature generation and verification for AVR microcontrollers, which are efficient, timing analysis resistant through constant time implementation, and SPA-protected by the randomized coordinates countermeasure. Our EdDSA-Ed25519-SHA512 implementation improves the current state of the art [27] performance for signing in 17.2%, with 19 221 517 cycles (constant time, randomized coordinates, and with a lookup-protected precomputed table of 8 points), and for verification in 5.7%, with 30 776 942 cycles. We also test the SPA leakage of our constant time implementation of the Montgomery Ladder scalar multiplication algorithm for Curve25519 with the randomized coordinates countermeasure by

[1] They used a version of the binary Extended Euclidean Algorithm for field inversion which is not constant time, and no SPA-specific countermeasure was applied.

running CRI's test vector leakage assessment methodology (TVLA) [22,50]. To the best of our knowledge, this is the first work to provide a SPA leakage assessment of an implementation of ECC on AVR. Finally, we also show that addresses of loads from Flash memory leak through power, and that such leakage can be exploited by template SPA.

2 Side-Channel Analysis on the AVR

2.1 Timing Analysis

Timing attacks against implementations of cryptographic algorithms exploit the fact that the elapsed time typically varies and depends on the specific value of the input data being processed on the particular run, for fixed (e.g., key) or variable (e.g., plaintext or ciphertext) data. Vulnerability to timing analysis implies vulnerability to power analysis, as time differences can be visually detected in power traces. The following are recommendations to prevent timing analysis [7].

Avoiding Secret-Dependent Load Addresses. This is necessary when the architecture has a memory hierarchy. It is not the case of AVR architecture, which has just one memory level, the SRAM, and in which all accesses take the same time. Thus, we do not implement this recommendation.

Avoiding Secret-Dependent Branch Conditions. In other words, to avoid data flow from secret data to branch conditions. In the case of AVR, there is no branch prediction mechanism, so this problem could be solved by balancing the number of instructions executed in the two conditions of the branch, but it has to be done at the assembly level, is tedious and error prone. Instead, we solve it by using conditional move operations implemented with logical operations.

2.2 Simple Power Analysis

Generally speaking, power analysis exploits the fact that the instantaneous power consumption of a device depends on both the data processed and the operation performed [37,34]. Power analysis attacks are classified as SPA even if the attacker needs to obtain more than one trace to succeed, maybe from different input data values, provided statistical analysis of the traces are not required [37].

Power analysis countermeasures for both SPA and DPA are based on the reduction or elimination of the dependency between the power consumption of a cryptographic device and the intermediate values used by the algorithm, and are classified in two main groups: hiding and masking [37]. In this work we apply both kinds of countermeasures. Hiding is employed through highly regular[2] [30] scalar multiplication algorithms, which also do not assume that

[2] An algorithm is said to be highly regular when: (i) it always executes the same instructions, in the same order, for all possible input values; and (ii) there is no dummy instructions, i.e., all instructions are effective.

distinct operations have the same leakage characteristics[3]. Masking is applied by randomizing the point coordinate representation, to protect against SPA and template SPA [39].

3 Prime Field Arithmetic

Curve25519 and Ed25519. Curve25519 [5] is the Montgomery curve $E(\mathbb{F}_p)$: $y^2 = x^3 + 48662x^2 + x$ over the prime field $\mathbb{F}_p$, $p = 2^{255} - 19$. Ed25519 [6] is a twisted Edwards curve birationally equivalent to Curve25519, defined by $E'(\mathbb{F}_p) : -x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2$ over the same prime field $\mathbb{F}_p$.

Prime field $\mathbb{F}_{2^{255}-19}$. Following the representation proposed in [9], we also represent an element of $\mathbb{F}_p$ as an integer modulo $2^{256}-38$ during field operations, as do previous implementations of Curve25519 and Ed25519 in AVR [27,19]. This redundant representation allows for a more efficient reduction than reducing directly modulo p. Only in the end of the scalar multiplication calculation, if the integer is not already in $\mathbb{F}_p$, we subtract p in constant time.

Field Multiplication and Squaring. Multi-precision multiplication (256-bit) is implemented as a 3-level subtractive Karatsuba [10,28]. This variant of Karatsuba avoids the carry bits when computing the middle partial product, but it requires the computation of two absolute differences of the low and high halves of the operands, $|A_L - A_H|$ and $|B_L - B_H|$, and one conditional negation of the product of these differences. The bottom 32-bit multiplier is fully unrolled. Field squaring is implemented as a 3-level subtractive Karatsuba, in which case there is no conditional negation of M. The 32-bit multiplier from the multiplication is reused here, through a function call, at the bottom level. Both operations are implemented in assembly, are branch-free and partially unrolled.

Multiplication by Constant 121666. This multiplication is required for the group arithmetic in Curve25519. Since the constant representation requires 3 bytes ($1 \parallel \texttt{DB} \parallel 42$), multiplying it by a single word takes only 2 multiplications and a few addition instructions.

Field Inversion. We use Fermat's theorem, $x^{-1} \equiv x^{p-2} \pmod{p}$, to compute inversion in $\mathbb{F}_p$ in constant time. We use the same addition chain as [5,27], consisting of 254 squares and 11 multiplications, but we reduce the number of temporary field variables required from 10 to only 5.

4 Arithmetic Modulo Ed25519 Group Order

EdDSA-Ed25519 signature scheme requires addition and multiplication modulo the Ed25519 group order (N). We implemented reduction modulo N in C using a constant time version of the Barret algorithm obtained by unrolling the final

[3] For example, it is not supposed that field squaring and multiplication exhibits the same leakage patterns.

subtraction loop into two copies of its body (the maximum number of iterations) and using conditional moves implemented in constant time.We precomputed the reciprocal of the modulus, $R = \lfloor b^{2n}/N \rfloor = \lfloor 256^{64}/N \rfloor$, a parameter of the Barret algorithm, and stored it in program memory. The multiplication calls the 256-bit multiplier and then reduces fully. The addition also reduces fully and is implemented in assembly.

5 Scalar Multiplication

The most computationally expensive operation in ECC is the scalar multiplication (ECSM), also known as point multiplication (by a scalar). Protocols usually involve three cases: fixed base point (kG), where G is a fixed point (usually the subgroup generator) and k is a scalar; variable base point (kP), where P is a point not known in advance; and the double scalar multiplication ($kP + sG$), where P is variable and G is fixed.

Several algorithms are available for variable, fixed and double-base scalar multiplication on Curve25519 and Ed25519. The major criteria we used for the selection of ECSM algorithms were high regularity [30], followed by performance. In the case of fixed-base and double-base ECSM, the size of the table of precomputed points was also an important criteria, in view of the small Flash memory space on ATmega328P. We also wanted to explore the performance of SPA-safe ECSM algorithms that, to the best of our knowledge, were not yet implemented in AVR microcontrollers, such as the FLS fixed-base ECSM algorithm [20,8].

5.1 Extended Twisted Edwards Coordinates

The most efficient formulas for point arithmetic on twisted Edwards curves were proposed by Hisil et al [25], representing points in the extended twisted Edwards coordinates: a point $P = (x, y)$ is represented by the quadruple $(X : Y : T : Z)$, such that $x = X/Z$, $y = Y/Z$, $xy = T/Z$ and $Z \neq 0$. The auxiliary coordinate T augments homogeneous projective coordinates $(X : Y : Z)$ with the product of x and y, and has the property $T = XY/Z$. The group identity element is represented by $(0 : 1 : 0 : 1)$, the negative of an element $(X : Y : T : Z)$ is $(-X : Y : -T : Z)$. A point in affine coordinates (x, y) can be converted to extended twisted Edwards coordinates by $X = x$, $Y = y$, $T = xy$ and $Z = 1$. To convert back to affine, T is ignored and an inversion and two multiplications are required: $x = X/Z$ and $y = Y/Z$. Similarly, a point can be converted to homogeneous projective coordinates $(X : Y : Z)$ simply by discarding T.

5.2 Variable-Base Scalar Multiplication

Montgomery Ladder Algorithm. Our implementation uses the formulas introduced by Montgomery [41] for efficient x-coordinate differential point addition and doubling on elliptic curves in the Montgomery form, as do previous

implementations of Curve25519-ECDH [7,27]. The so-called Montgomery ladder algorithm comprises a sequence of 255 steps, known as *ladder steps*, each performing one point addition and one point doubling, where a point is represented by projective coordinates $(X : Z)$, where $x = X/Z$ is the respective affine x-coordinate. For a high level description of the algorithm, we refer the reader to [19, §2]. Our implementation conditionally swaps the two point variables, $P1 = (X_1 : Y_1)$ and $P2 = (X_2 : Y_2)$, in constant time, before the point operations in each ladder step. Point addition requires $3M + 2S$ and point doubling requires $2M + 2S + 1M_c$ [4].

Joye's Double-Add Algorithm. Joye's double-add [29] (Algorithm 1) is a variable-base right-to-left scalar multiplication algorithm, with no known SSCA attack, which always repeats the same pattern of effective operations: a point doubling is always followed by a point addition. The first operand (R_{1-b}) in the point addition is the result from the last point doubling, while the second operand is the result from a previous addition, not necessarily the last one. For this reason, we use the following coordinate systems in the point operations:

$$\text{ExtTwistEd} := 2 \cdot \text{HomoProj}$$
$$\text{ExtTwistEd} := \text{ExtTwistEd} + \text{ExtTwistEd}$$

The point doubling algorithm (Algorithm 2 in Appendix B) is based on the dedicated doubling formula from [25, §3.3], is optimized for $a = -1$ (the case for Ed25519) and costs $4M + 4S$. In its implementation, the input point is actually represented in twisted Edwards coordinates, but is then converted to homogeneous coordinates simply by ignoring the T coordinate (see Sect. 5.1). The point addition algorithm (Algorithm 3 in Appendix B) is based in the unified and complete point addition formula from Hisil et al [25, §3.1] and is optimized for the case $a = -1$. It costs $8M + 1M_c$, but as the constant is large in this case, a full multiplication is needed, therefore the effective cost is $9M$. The scalar multiplication cost is thus $255 \cdot (13M + 4S)$.

Goundar *et al*'s Signed Digit Algorithm. In order to prevent SPA-type attacks, Goundar *et al* [24] proposed the use of the *zeroless signed-digit expansion* (ZSD) in the binary left-to-right or right-to-left algorithms. The odd scalar k is recoded on-the-fly with digits in the set $\{-1, 1\}$ (Algorithm 4 in Appendix B). We use extended twisted Edwards coordinates for point addition and doubling. The point addition is actually a readdition, because the second operand, $R_1 = (X_2 : Y_2 : T_2 : Z_2)$, can only be P or $-P$, so we can cache the result of kT_2, $(Y_2 - X_2)$, $(Y_2 + X_2)$ and $2Z_2$, saving a multiplication. Therefore, the scalar multiplication costs $254 \cdot (12M + 4S)$.

5.3 Flash Memory Address Leakage Through Power

When the base point P is fixed, scalar multiplication algorithms can employ (offline) precomputation involving P to speedup the (online) evaluation phase

[4] M_c means multiplication by a constant, 121666 in this instance.

of the scalar multiplication. For that end, a table of multiples of the base point is typically precomputed and stored in a non-volatile memory. In the evaluation phase, the points in the table are looked up based on some indexing method, whose index values are dependent on bits of the (secret) scalar.

In the case of AVR, Flash memory is used to store the precomputed point table. The time required to load a word from the Flash to the SRAM is constant, independent of the address referenced. However, different index values correspond to different Flash addresses, with possibly distinct Hamming weights, and therefore potentially distinct power consumption characteristics. We designed an experiment to evaluate whether this kind of leakage occurs and whether the leakage level is sufficient to distinguish between all the possible Hamming weights of the addresses referenced in Flash reads. On AVR, Flash has a different address space than SRAM, and are 16-bit on the ATmega328p.

The experiment consisted of selecting a set of addresses whose Hamming weights are in the set $S_{HW} = \{2, ..., 8\}$. We selected 7 addresses in the range from `0x00A7` to `0x00FF`, which have the upper 8 bits zeroed, one for each Hamming weight in S_{HW}. Let S_{addr} be this set[5]. We wrote a fixed byte value (`0xDE`) to all addresses in this set. We then executed reads from this addresses (`LPM` instruction) to a (fixed) register (the byte `0x00` is written to this register in advance). For each address Hamming weight in S_{HW}, 100 traces were captured. Each power trace captured consists of the power consumption of a sequence of instructions including the target `LPM`. The samples corresponding to the `LPM` instruction were visually identified, and then, for each address Hamming weight, the average of the corresponding traces were computed.

Figure 1a shows the "average traces" for each address Hamming weight, for the sample points corresponding to the `LPM` instruction. We can see that the voltage values for the sample index 98 are the best single-point distinguishers for the address Hamming weights in the set S_{HW}. Therefore, we select the average and sample standard deviation of voltages at this point to analyze whether this leakage can be exploited to recover the Hamming weights.

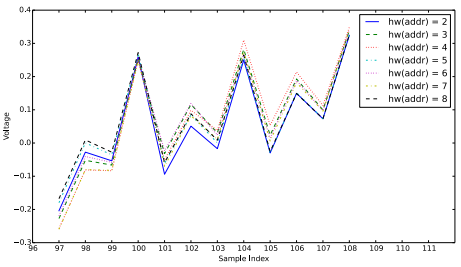
Figure 1b shows the average voltage and the 95% confidence interval for each Hamming weight in the set S_{HW}. We cannot classify with strong statistical significance the Hamming weight of a Flash address based only in one voltage sample, because every confidence interval overlaps with at least one other confidence interval[6]. However, the following groups can be distinguished: $\{2\}$, $\{3, 6\}$, $\{4, 7\}$, $\{8, 5\}$. This enables an adversary to detect sets of possible addresses of the points being loaded from Flash to SRAM. In practice, in the context of fixed-base scalar multiplication, during the lookup of a point stored in the precomputed point table an adversary could measure the address leakage from the

[5] We selected this range of Flash addresses, because it is available for user programs, but other ranges could be used. The following were the addresses selected: `0x00c0` (hw=2), `0x00e0` (hw=3), `0x00f0` (hw=4), `0x00f8` (hw=5), `0x00fc` (hw=6), `0x00fe` (hw=7) and `0x00ff` (hw=8).

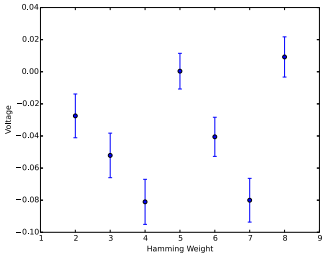
[6] In the case of Hamming weights 4 and 5, the confidence interval for 4 contains the one for 7.

load of each word of each coordinate of the point, and thus could combine the leakage values during the single point lookup to uniquely determine its index.

To counteract this kind of leakage, we implemented a constant time table lookup. For each point table lookup, we read all words in the point table and use bitwise arithmetic such that in the end the words of the coordinates values of the point requested are in the target buffer in SRAM (see Algorithm 6 in Appendix B).



(a) Voltage versus Sample index.



(b) Voltage versus Hamming weight.

Fig. 1. Figure 1a shows the Voltage versus Sample index, for each Flash address Hamming weight from 2 to 8, in the interval of samples/points in the average trace corresponding to the LPM instruction. LPM takes 3 CPU cycles, and the sampling rate is 4 samples per cycle. Figure 1b shows the Voltage versus Hamming weight for the second sample (sample index 98), with 95% confidence interval bars, for LPM instructions referencing Flash addresses with different Hamming weights.

5.4 Fixed-Base ECSM for Ed25519 Key Generation and Signing

Modified LSB-set Algorithm (FLS). We implemented Faz-Hernandez et al's modified LSB-set comb algorithm [20], henceforward named FLS, according to the specification in [8, Alg. 7]. Beyond the curve-related parameters, which have already been fixed by the curve selection, different pairs of values for the number of tables of precomputed points v and the window width w have been selected and experimentally evaluated, to determine their actual performance when protection against Flash memory address leakage is applied, and also to compare the required storage costs. Points are represented in extended twisted Edwards coordinates, both in the working variables and in the precomputed table, and point addition and doubling formulas are the same used for Joye's (Algorithms 3 and 2 in Appendix B). We evaluated the performance with 4 precomputed points (1KB), $(v = 1, w = 3)$ (1 table) and $(v = 2, w = 2)$ (2 tables), and with 8 points (2KB), $(v = 1, w = 4)$ (1 table) and $(v = 2, w = 3)$ (2 tables). We also evaluated the performance impact of the table lookup protection. In the case of two tables, we use a similar technique as that for a single table (Algorithm 6 in Appendix B), to guarantee that all words are read from both tables, but only the desired point is left in the destination variable.

5.5 Projective Coordinate Randomization

In a template SPA [39] attack the adversary first characterizes the power consumption of a sequence of instructions executed on a device similar to the target device, when a fixed pair (key, data) is processed, and repeats the process for several such pairs, resulting in a set of power consumption templates, one for each pair (template building phase). After that, the templates are matched against a single trace captured from the target (template matching phase).

Highly regular scalar multiplication algorithms implemented in constant time are not enough to protect against template SPA attacks, as it was shown in [39,4]. According to Medwed and Oswald [39], the only way to make an implementation resistant to template SPA attacks is to make it resistant against DPA attacks, and they also assert that among Coron's proposed countermeasures for DPA in ECC [16], only randomized projective coordinates can prevent template SPA.

We applied the randomized projective coordinates countermeasure to the projective coordinates $(X : Z)$ in Montgomery Ladder and the extended twisted Edwards coordinates $(X : Y : T : Z)$ in Joye's Double-Add, Goundar's Signed-digit and FLS algorithms. In the case of Montgomery Ladder, we generate random $\lambda \in \mathbb{F}_p \backslash \{0\}$ in the beginning of the algorithm and do $Z_2 \leftarrow \lambda$ and $X_2 \leftarrow u \cdot \lambda$, where u is the x-coordinate of the input point P and $P_2 = (X_2 : Z_2)$ is the second point variable. In the case of Joye's Double-add and Goundar's Signed-digit algorithms, we randomly generate $\lambda \in \mathbb{F}_p \backslash \{0\}$ and do $X' \leftarrow \lambda x$, $Y' \leftarrow \lambda y$, $T' \leftarrow xY'$ and $Z' \leftarrow \lambda$, where $P = (x, y)$ is the input point in affine coordinates and the resultant point $P' = (X' : Y', T', Z')$ is used in place of P in the remainder of the algorithms.

In FLS algorithm, we randomize the coordinates of the first point loaded from the table of precomputed points, $P_0 = (X : Y : T : Z)$, as follows: generate random $\lambda \in \mathbb{F}_p \backslash \{0\}$ and do $X' \leftarrow \lambda X$, $Y' \leftarrow \lambda Y$, $T' \leftarrow \lambda T$ and $Z' \leftarrow \lambda Z$. The resultant point $P'_0 = (X' : Y' : T' : Z')$ is used in place of P_0. When this countermeasure is applied, the values of the coordinates of the accumulator point Q are randomized, changing from one execution of the scalar multiplication to the other, because the value of P'_0 is assigned to Q in the beginning of evaluation stage [8, Alg. 7]. Furthermore, as an additional measure to protect against template SPA attacks targeting the loading of words stored Flash (i.e., the value of the words itself, rather than their addresses), the extended twisted Edwards coordinates of each point $P_{i,j}$ are randomized before being stored in the table, with a random λ generated for each point, in the same way P_0 was randomized.

6 Hashing and PRNG

SHA-512. The original EdDSA-Ed25519 [6] and the AVRNaCl [27] implementations of EdDSA-Ed25519 have chosen SHA-512 [42] as the hash function, therefore we have also selected it to be able to compare our results to theirs. Our goal is to achieve a small and simple implementation, so we decided to implement in assembly only the low-level functions (`add64`, `and64`, `or64`, `xor64`, `rotr64` and

`shr64`), while the higher level ones are implemented in C and optimized for size. The constant `H0` and round constants were kept in program memory.

Hash_DRBG. Random numbers are required for the projective coordinates randomization and for key pair generation on the device, in particular if ephemeral ECDH (ECDHE) is used. For this purpose, we implemented the Hash_DRBG pseudorandom number generator [3] at the 128-bit security level, with SHA-512 as the underlying hash function.[7] The PRNG is seeded during instantiation, and its seed is the output of a derivation function whose inputs are the entropy input and nonce. The nonce is stored in the EEPROM when the device is programmed. The entropy input is also assumed to be stored in the EEPROM, as we do not implement entropy gathering from physical sources.[8]

7 Elliptic Curve Protocols

Elliptic Curve Diffie-Hellman with Curve25519 (ECDH-Curve25519). ECDH-Curve25519 protocol consists of two operations: generate key pair and compute shared secret. The latter consists mainly of a variable-base scalar multiplication, which is implemented using the Montgomery Ladder, and an inversion in $\mathbb{F}_p$. The first requires a scalar multiplication by a fixed base point, also implemented with the Montgomery Ladder.

Edwards Digital Signature Algorithm with Ed25519 (EdDSA-Ed25519-SHA512). Key generation consists of a scalar multiplication by the subgroup generator and a SHA512 hash. The signature generation applies SHA512, a fixed-base scalar multiplication, addition and multiplication modulo the subgroup order, and simple point and scalar encoding operations. The signature verification consists of simple point and scalar decoding operations followed by SHA512 and a double scalar multiplication, where one of the points is fixed (subgroup generator) and the other is variable (the signer public key). The latter is implemented with "Shamir's trick", a special case of Straus's algorithm [49].

8 Benchmarking Results

The source code was compiled with AVR-GCC v4.8.2, the size optimization `-Os` and `-fomit-frame-pointer` options were applied to the C sources, and the program was linked with global linker optimization `-flto`. Table 1 shows the benchmarking results. The signature generation uses the FLS algorithm with

[7] Faster approved hash functions, such as SHA-256 or even SHA-1, could be used at this security level, but we decided to just call the SHA-512 function already available to not increase the code size.

[8] One such scheme proposed in the literature is Hlavac et al's [26] method of generating true random numbers on the AVR based on the jitter of the built-in RC oscillator, requiring only an external oscillator. However, the resulting TRNG is slow, being capable of generating only 8 bits of entropy per second.

($v = 1, w = 4$) (8 points, 1 table), with and without the table lookup protection and randomized coordinates countermeasures. In the results for the functions protected with the coordinate randomization, the PRNG overhead is not taken into account, i.e., the required number of random bytes are readily available.

The results of our implementation of EdDSA-Ed25519-SHA512 improve the state of the art performance [27], requiring 19 047 706 cycles for signing, an improvement of 17.9%, and 30 776 942 cycles for verification, an improvement of 5.7%. The overhead of the countermeasures table lookup protection and randomized projective coordinates to the FLS algorithm is only 1.0%. Similarly, when these countermeasures are applied on the signature generation function, the overhead is also very small (0.9%). In the case of the compute shared secret function, the overhead of the coordinate randomization is only 0.04%.

Despite having a slower field multiplication than Hutter and Schwabe implementation (6208 cycles) [27], we implemented a dedicated field squaring algorithm in assembly while the authors simply reused the multiplication function for squaring, resulting in faster scalar multiplications for both Curve25519 and Ed25519. Dull et al [19] described an efficient implementation of field multiplication and squaring for AVR using more efficient algorithms, significantly improving the state-of-the-art performance with 13 900 397 cycles for computing a ECDH-Curve25519 shared secret key.

9 Timing and Simple Power Analysis Leakage Evaluation

Side-channel security evaluations of cryptography devices comprise two phases: *measurement* and *analysis*. The output of such an evaluation should be an assertion, indicating whether the device is vulnerable (*Fail*) or not (*Pass*), given the constraints of the evaluation process[9]. The proper measurement of the side-channel traces and its limitations must be properly accounted for, or else the analysis process could be undermined, probably resulting in false positives, or worse, false negatives.

Current evaluation methodologies (e.g. Common Criteria [17]) consist of performing a battery of known side-channel attacks against the device under test (DUT) in an attempt to recover the key. Nonetheless, the rapid evolving set of side-channel attacks proposed in the literature incur both a more demanding level of expertise of test operators and an increase on the evaluation time. Even when all attack attempts have failed, residual side-channel leakages may be available, which may reveal new attack paths for an adversary.

CRI proposed the Test Vector Leakage Assessment (TVLA) testing methodology, to solve the aforementioned issues, which is claimed to be effective, in the sense that it is reproducible and is a reliable indicator of the resistance achieved, and cost effective, meaning that "validating a moderate level of resistance (e.g., FIPS 140 level 3 or 4) should not require an excessive amount of testing time per algorithm or test operator skills" [22]. Their approach differs fundamentally from

[9] E.g., accuracy of the testing equipment, technical expertise and available time.

Table 1. Benchmarking results on ATmega328P.

Operation Class	Operation/Algorithm	Msg.(B)	Cycles	Stack(B)
Field Arith.	Field Multiplication	-	7555	-
	Field Squaring	-	5666	-
	Field Inversion	-	2 000 762	-
Group Order Arith.	Barret Reduction	-	43 045	-
	Group Order Addition	-	54 303	-
	Group Order Negation	-	46 773	-
	Group Order Multiplication	-	72 438	-
Variable-base ECSM Curve25519	Montgomery Ladder	-	20 153 658	-
	Montgomery Ladder, rand. coord.	-	20 161 213	-
Variable-base ECSM Ed25519	Joye's Double-Add	-	42 436 422	-
	Joye's Double-Add, rand. coord.	-	42 459 087	-
	Goundar's Signed-digit	-	35 757 016	-
	Goundar's Signed-digit, rand. coord.	-	35 779 681	-
Fixed-base ECSM Ed25519	FLS ($v = 1$, $w = 3$)	-	21 553 188	-
	FLS ($v = 2$, $w = 2$)	-	26 661 293	-
	FLS ($v = 1$, $w = 3$), lookup prot.	-	21 658 857	-
	FLS ($v = 1$, $w = 4$)	-	18 119 234	-
	FLS ($v = 2$, $w = 3$)	-	19 170 150	-
	FLS ($v = 1$, $w = 4$), lookup prot.	-	18 264 710	-
	FLS ($v = 1$, $w = 4$), lookup + rand. coord.	-	18 298 387	-
Double ECSM Ed25519	Shamir's trick	-	28 105 811	-
Hash	SHA-512	64	554 280	-
	SHA-512	1024	4 974 380	-
ECDH-Curve25519	Compute shared secret [27]	-	22 791 579	677
	Compute shared secret [19]	-	13 900 397	494
	Compute shared secret	-	20 254 426	686
	Compute shared secret, rand. coord.	-	20 261 981	743
EdDSA-Ed25519	Signature generation [27]	64	23 216 241	1642
	Signature generation	64	19 047 706	1473
	Signature generation, lookup + rand. coord.	64	19 221 517	1511
	Signature verification [27]	64	32 634 713	1315
	Signature verification	64	30 776 942	1226

the attack-focused evaluation strategies currently employed, taking a black-box and detection-focused strategy [38].

The measurement phase of TVLA is based on the collection of side-channel traces when standardized test vectors are provided as input to the algorithm being tested, and establishes requirements for power measurement equipment and setup, data collection, signal alignment and preprocessing. The analysis phase is based on Welch's t-test, can detect different types of leakages and allows the analyst to identify points in time that deserve further investigation. The testing methodology has so far been applied to AES and RSA implementations[10].

Other methodologies, based on continuous [13] and discrete [12] mutual information have also been proposed. Oswald et al [38] analyzed methodologies [22], [12]

[10] For AES, by the methodology authors [22,15] and independently [38]; and for RSA software implementations [50].

and [13], and concluded they have similar statistical power. The recent work of Schneider and Moradi [48] address how to perform the t-test in [22] at higher orders, and how to extend it to multivariate settings.

9.1 Application of CRI's Methodology to ECC

We apply CRI's methodology to our implementation of ECDH-Curve25519, using Chipwhisperer as the power measurement equipment. Specifically, we select a set of test vectors (Table 2) to be used for the power measurement phase, which cover normal and special cases of the field and group arithmetic when implemented using the chosen algorithms. Table 3 shows categories of special values used in Sets 4 and 5 for the compute shared secret function.

Table 2. Sets of test vectors for SPA leakage analysis (k is the secret scalar and P is the point).

Set #	Properties	Rationale
1	constant k, constant P	This is the baseline. The tests compare power consumption from the other sets against it.
2	constant k, varying P	Goal is to detect systematic relationships between power consumption and the P value.
3	varying k, constant P	Goal is to detect systematic relationships between power consumption and the k value.
4	constant k, special P	Edge cases of the algorithms used.
5	special k, constant P	Edge cases of the algorithms used.

Table 3. Categories of special values for n and q in ECDH-Curve25519 compute shared secret function (q is the encoded point, n is the encoded scalar and l is the subgroup order).

Cat. #	Properties
1	$q \in \{0, 1, ..., 1023\}$
2	$q \in \{p_{25519} - 1, ..., p_{25519} - 1024\}$
3	$n \in \{0, ..., 1023\}$
4	$n \in \{l - 1, ..., l - 1024\}$
5	q has a low Hamming Weight (≤ 230)
6	q has a high Hamming Weight (≥ 25)

9.2 Measurement Setup and Capture of Power Traces

Time Measurement Setup. In an AVR CPU, as the clock frequency is constant, the elapsed time of an algorithm can be measured by simply counting the number of cycles it takes to execute. We use timer interrupts which increment a 16-bit and a 8-bit counter, resulting in 24-bit resolution.

Power Measurement Setup. In Chipwhisperer, power consumption traces are captured by the OpenADC board, which features an ADC with a sample rate of 105 MS/s, 10-bit sample resolution, 120 MHz analog bandwidth and 0 to 55 dB gain (software adjustable). The sample buffer capacity is 24k samples. We used the following parameters values in our setup, adhering to the minimum requirements from [22]. ADC frequency is set to 29.5 MHz, which is exactly four times the clock rate provided to the ATMega328P (4 x 7.37 MHz), the analog gain is set to 40 dB. The trigger mode is configured to rising clock edge.

Capture of Power Traces. The first issue we faced to capture power traces using Chipwhisperer was the small size of the samples buffer. In the latest version available of its FPGA bitstream and capture software, the samples buffer is implemented as a FIFO using cells of the small Spartan 6 LX25 FPGA, limiting its size to just 24573 samples, 10 bits each[11]. In our settings, this corresponds to a limit of 6143 cycles (4 samples per cycle) per acquisition operation.

Assuming a Curve25519 variable-base scalar multiplication takes $20M$ cycles, then 3256 acquisitions would be required to cover the whole operation. If such acquisitions are made sequentially, however, samples are lost in the time interval between an acquisition and the next, creating a gap in the trace, because when the buffer is full, its content must be sent to the host computer through serial connection before it could be emptied. To work around this limitation, we captured 6 traces (numbered from 0 to 5) of the full scalar multiplication, each one with gaps. Even and odd-numbered traces have time offsets such that the gaps in even traces don't overlap the gaps in odd traces. We compute the average of the 6 traces, not including the gaps in the average computation, obtaining an *average trace* to be used as a single trace in the analysis phase.

We obtained 200 average traces for each test vector set, for a total of 1000 traces. The complete trace capture process, including averaging, compression and disk storage took around 6 hours. Each uncompressed trace has $80,614,632$ samples, and occupies $100,768,290$ bytes. The sample acquisition process begin with a trigger produced by the code running in the AVR, through the assertion of bit 0 of `PORTC` register, just before the call to the scalar multiplication function. Similarly, the end of the acquisition was triggered by deasserting the same bit after the function returns. As this method provides a very precise synchronization of the start and end times of capture across traces, no trace alignment was needed.

9.3 SPA Leakage Analysis

The leakage analysis phase of our implementation of ECDH-Curve25519 with randomized coordinates countermeasure is identical to CRI's TVLA [50], and is conducted in the following way. Let $\{DS_1, \ldots, DS_5\}$ be the sets of power traces corresponding to the selected test vectors sets. The full test consists of running the (pairwise) tests described in [50] for each of the following pairs of datasets:

[11] The board provides a 1GB DDR memory, but it is not supported yet.

$\{(DS_1, DS_2), \ldots, (DS_1, DS_5)\}$. If any of the previous tests fails, then the top-level test fails and the implementation is deemed to have FAILED. Otherwise, it PASSED. We chose the confidence threshold $C = 4.5$, the same value used in CRI's methodology for RSA [50].

10 Side-Channel Analysis Results

Timing Analysis. Timing measurements were obtained for ECDH-Curve25519 compute shared secret function using a superset of the test vectors used in the SPA analysis, which includes additional randomly generated test vectors. For the timing measurements of EdDSA-Ed25519-SHA512 signature generation, we used randomly generated test vectors and those covering corner cases. The results obtained show that the implementations of both functions are constant time, with respect to the private key value.

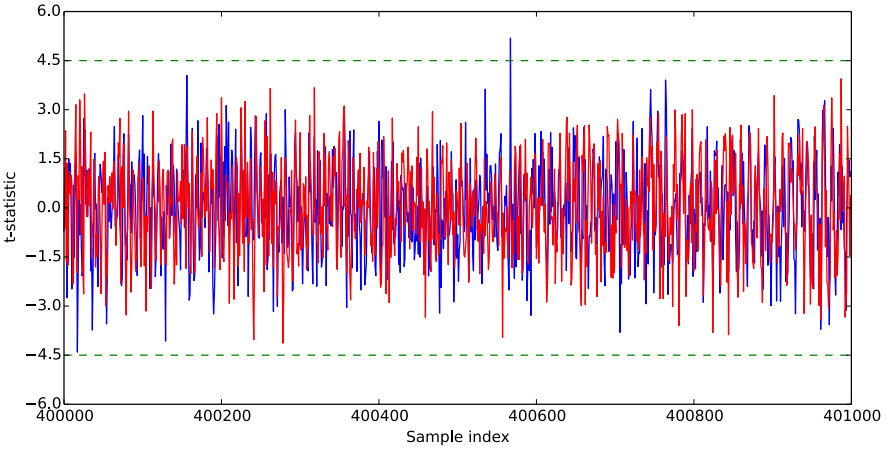


Fig. 2. t-statistic versus sample index for the experiment comparing DS_1 and DS_3, for two independent groups of traces; group A (blue) and group B (red).

SPA Analysis. The leakage analysis methodology was applied to our implementation of ECDH-Curve25519 with randomized coordinates. Figure 2 shows the t-statistic for a small range of sample indices[12] (time instants), for one run of Welch's t-test for group A $(S_{A,1}, S_{A,2})$ of vectors selected from DS_1 and DS_3, and the same test run over the independent group B $(S_{B,1}, S_{B,2})$.[13] The t-statistic for group A is above $C = 4.5$ at one time instant, meaning a possible strong dependence between power consumption and key value at that instant. But, as

[12] This time interval was selected because it illustrates a range where the t-statistic values are relatively high, compared to other time instants.

[13] Groups A and B are a partition of test vector sets DS_1 and DS_3: $(S_{A,1} \subset DS_1, \ S_{A,2} \subset DS_3)$ and $(S_{B,1} = DS_1 \setminus S_{A,1}, \ S_{B,2} = DS_3 \setminus S_{A,2})$.

it did not occur at the same time and in the same direction for group B, it is therefore considered a false positive by the methodology and thus discarded. The test results for each pair of test vector sets $\{(DS_1, DS_2), \ldots, (DS_1, DS_5)\}$ showed that at a few time instants the t-statistic value for one of the groups is above 4.5 or below -4.5, but not for both groups at the same time. Therefore, we can conclude that our SPA protected implementation of ECDH-Curve25519 passed the SPA leakage evaluation.

11 Conclusion

We describe an efficient implementation of protocols ECDH-Curve25519 and EdDSA-Ed25519-SHA512 for an AVR 8-bit microcontroller. The implementations prevent timing attacks by using regular algorithms and constant time field arithmetic. SPA and template SPA protection is provided by the use of coordinate randomization and by avoiding secret-address dependent loads from Flash memory. The effectiveness of the SPA countermeasures for ECDH-Curve25519 is evaluated through the application of CRI's TVLA methodology with selected test vectors, using Chipwhisperer.

We identify the following open problems. Consider the operations in this excerpt, which are typically used in constant time code (e.g., CSWAP and CMOV) to avoid branches dependent on secret data: $\text{MASK} \leftarrow -b$; $t \leftarrow \text{MASK} \wedge x$. In this example, $b \in \{0, 1\}$ is a secret dependent bit and, therefore $\text{MASK} \in \{0, 255\}$, in 8-bit unsigned integer representation. We notice that the difference in Hamming weight between the possible mask values are maximal (7), meaning that if the device leaks the Hamming weight of the (sum) of the AND instruction operands or their Hamming distance, and the adversary knows the value of x, than she may be able to exploit it for a template SPA attack. Also, according to a survey of SCA in ECC [18], some known SCA attacks are not addressed by the countermeasures we used, such as RPA [23] and ZPA [1], and thus deserve an analysis regarding their applicability to Curve25519 and Ed25519. Another further work is to analyze how strong is the leakage of the data read from the Flash and find a simple leakage model that best characterizes it.

References

1. Akishita, T., Takagi, T.: Zero-value point attacks on elliptic curve cryptosystem. In: Boyd, C., Mao, W. (eds.) ISC 2003. LNCS, vol. 2851, pp. 218–233. Springer, Heidelberg (2003)
2. ANSSI. Mécanismes cryptographiques - Règles et recommandations. Technical report, Agence nationale de la sécurité des systèmes d'information (2014)
3. Barke, E., Kelsey, J.: SP 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators. Technical report, NIST (2012)
4. Batina, L., Chmielewski, L., Papachristodoulou, L., Schwabe, P., Tunstall, M.: Online Template Attacks. In: Meier, W., Mukhopadhyay, D. (eds.) INDOCRYPT 2014. LNCS, vol. 8885, pp. 21–36. Springer, Heidelberg (2014)

5. Bernstein, D.J.: Curve25519: new diffie-hellman speed records. Technical report, University of Illinois at Chicago (2006)
6. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.-Y.: High-speed high-security signatures. Journal of Cryptographic Engineering 2(2), 77–89 (2012)
7. Bernstein, D.J., Lange, T., Schwabe, P.: The security impact of a new cryptographic library. In: Hevia, A., Neven, G. (eds.) LatinCrypt 2012. LNCS, vol. 7533, pp. 159–176. Springer, Heidelberg (2012)
8. Bos, J., Costello, C., Longa, P., Naehrig, M.: Selecting elliptic curves for cryptography: an efficiency and security analysis. Journal of Cryptographic Engineering, 1–28 (2015)
9. Bos, J.W.: High-performance modular multiplication on the cell processor. In: Hasan, M.A., Helleseth, T. (eds.) WAIFI 2010. LNCS, vol. 6087, pp. 7–24. Springer, Heidelberg (2010)
10. Brent, R.P., Zimmermann, P.: Modern Computer Arithmetic. Cambridge University Press (2010)
11. BSI. Algorithms for Qualified Electronic Signatures. Technical report, Bundesamt für Sicherheit in der Informationstechnik (2014)
12. Chatzikokolakis, K., Chothia, T., Guha, A.: Statistical Measurement of Information Leakage. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 390–404. Springer, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-12002-2_33, doi:10.1007/978-3-642-12002-2_33
13. Chothia, T., Guha, A.: A statistical test for information leaks using continuous mutual information. In: Proceedings - IEEE Computer Security Foundations Symposium, pp. 177–190 (2011)
14. Chu, D., Großschädl, J., Liu, Z., Müller, V., Zhang, Y.: Twisted Edwards-form elliptic curve cryptography for 8-bit AVR-based sensor nodes. In: Proceedings of the First ACM Workshop on Asia Public-Key Cryptography, pp. 39–44. ACM (2013)
15. Cooper, J., Demulder, E., Goodwill, G., Jaffe, J., Kenworthy, G.: Test Vector Leakage Assessment (TVLA) methodology in practice (Extended Abstract). Technical report, Cryptography Research Inc. (2013)
16. Coron, J.-S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999)
17. Criteria, C.: Common Criteria v3.1. Technical report, Common Criteria (2014)
18. Danger, J.-L., Guilley, S., Hoogvorst, P., Murdica, C., Naccache, D.: A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards. Journal of Cryptographic Engineering, 1–25 (2013)
19. Düll, M., Haase, B., Hinterwälder, G., Hutter, M., Paar, C., Sánchez, A.H., Schwabe, P.: High-speed curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers. In: Designs, Codes and Cryptography, pp. 1–22 (2015)
20. Faz-Hernández, A., Longa, P., Sánchez, A.H.: Efficient and secure algorithms for GLV-based scalar multiplication and their implementation on GLV-GLS curves. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 1–27. Springer, Heidelberg (2014)
21. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
22. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side channel resistance validation. Technical report, CRI (2011)

23. Goubin, L.: A refined power-analysis attack on elliptic curve cryptosystems. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 199–210. Springer, Heidelberg (2002)
24. Goundar, R., Joye, M., Miyaji, A., Rivain, M., Venelli, A.: Scalar multiplication on weierstraß elliptic curves from co-z arithmetic. Journal of Cryptographic Engineering 1(2), 161–176 (2011)
25. Hisil, H., Wong, K.K.-H., Carter, G., Dawson, E.: Twisted edwards curves revisited. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 326–343. Springer, Heidelberg (2008)
26. Hlavac, J., Lorencz, R., Hadacek, M.: True random number generation on an Atmel AVR microcontroller. In: 2010 2nd International Conference on Computer Engineering and Technology (ICCET), vol. 2, pp. V2–493–V2–495 (2010)
27. Hutter, M., Schwabe, P.: Nacl on 8-bit avr microcontrollers. In: Youssef, A., Nitaj, A., Hassanien, A.E. (eds.) AFRICACRYPT 2013. LNCS, vol. 7918, pp. 156–172. Springer, Heidelberg (2013)
28. Hutter, M., Schwabe, P.: Multiprecision multiplication on avr revisited. Journal of Cryptographic Engineering, 1–14 (2015)
29. Joye, M.: Highly regular right-to-left algorithms for scalar multiplication. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 135–147. Springer, Heidelberg (2007)
30. Joye, M.: Highly regular m-ary powering ladders. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 350–363. Springer, Heidelberg (2009)
31. Killmann, W., Lange, T., Lochter, M., Thumser, W., Wicke, G.: Minimum Requirements for Evaluating Side-Channel Attack Resistance of Elliptic Curve Implementations. Technical report, BSI (2011)
32. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation 48(177), 203–209 (1987)
33. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
34. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
35. Liu, Z., Großschädl, J., Wong, D.S.: Low-weight primes for lightweight elliptic curve cryptography on 8-bit AVR processors. In: Lin, D., Xu, S., Yung, M. (eds.) Inscrypt 2013. LNCS, vol. 8567, pp. 217–235. Springer, Heidelberg (2014)
36. Liu, Z., Wenger, E., Großschädl, J.: MoTE-ECC: Energy-scalable elliptic curve cryptography for wireless sensor networks. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 361–379. Springer, Heidelberg (2014)
37. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks: Revealing the secrets of smart cards, vol. 31. Springer (2007)
38. Mather, L., Oswald, E., Bandenburg, J., Wójcik, M.: Does My Device Leak Information? An *a priori* Statistical Power Analysis of Leakage Detection Tests. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 486–505. Springer, Heidelberg (2013)
39. Medwed, M., Oswald, E.: Template Attacks on ECDSA. In: Chung, K.-I., Sohn, K., Yung, M. (eds.) WISA 2008. LNCS, vol. 5379, pp. 14–27. Springer, Heidelberg (2009)
40. Miller, V.S.: Use of Elliptic Curves in Cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)

41. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. Mathematics of Computation 48(177), 243–264 (1987)
42. NIST. FIPS 180-2: Secure hash standard (SHS). Technical report, NIST (2001)
43. NIST. FIPS 140-3: Security Requirements for Cryptographic Modules. Technical report, NIST (2009)
44. NIST. SP 800-57 - Recommendation for Key Management. Technical report, National Institute for Standards and Technology (2012)
45. NSA. Fact Sheet Suite B Cryptography. Technical report, National Security Agency (2014)
46. O'Flynn, C., Chen, Z.D.: ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. In: Prouff, E. (ed.) COSADE 2014. LNCS, vol. 8622, pp. 243–260. Springer, Heidelberg (2014)
47. Quisquater, J.-J., Samyde, D.: ElectroMagnetic analysis (EMA): Measures and counter-measures for smart cards. In: Attali, S., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001)
48. Schneider, T., Moradi, A.: Leakage Assessment Methodology - a clear roadmap for side-channel evaluations. Cryptology ePrint Archive, Report 2015/207 (2015)
49. Straus, E.G.: Addition chains of vectors (problem 5125). In: American Mathematical Monthly, pp. 806–808 (1964)
50. Witteman, M., Jaffe, J., Rohatgi, P.: Efficient side channel testing for public key algorithms: RSA case study. Technical report, Cryptography Research (2011)

A ATmega328P Microcontroller and Chipwhisperer

The AVR is a family of 8-bit microcontrollers from Atmel featuring a RISC instruction set. It is a Harvard-based architecture with separate address spaces for data (SRAM), program (Flash) and non-volatile data (EEPROM). The ATmega328P has a 32KB Flash, a 2KB SRAM and a 1KB EEPROM. It has a maximum frequency of 20 MHz, but operates at 7.3728 MHz in Chipwhisperer. The register file contains 32 registers (R0-R31), among which 6 registers serve as pointers for indirect 16-bit addressing and have the following aliases: X (R27:26), Y (R29:R28) and Z (R31:R30). Arithmetic instructions take 1 cycle, with the exception of multiplication instructions, which take 2 cycles. Loads and stores from/to SRAM take 2 cycles. Loads from Flash memory take 3 cycles.

Chipwhisperer [46] is a toolbox consisting of open source hardware and software for side-channel analysis of AVR microcontroller software. It provides features for power and electromagnetic (SEMA and DEMA[14]) side-channel analysis, as well as clock and VCC glitching. On the hardware side, there is a capture board with an ADC and a Xilinx Spartan 6 FPGA, for system control and capture, and a target board with ATmega328P and XMega16A4A microcontrollers. Open source software for trace capture and analysis is also provided.

[14] Simple and Differential Electromagnetic Analysis, respectively.

B Algorithms

Algorithm 1. Joye's double-add right-to-left algorithm [29]

Input: Point $P \in E(\mathbb{F}_p)$ and $k = (k_{n-1}, \ldots, k_1, k_0)_2 \in \mathbb{N}$
Output: $Q = [k] \cdot P$
1: $R_0 \leftarrow P_\infty$, $R_1 \leftarrow P$
2: **for** i **from** 0 **to** $n-1$ **do**
3: $\quad b \leftarrow k_i$
4: $\quad R_{1-b} \leftarrow 2R_{1-b} + R_b$
5: **end for**
6: **return** R_0

Algorithm 2. Point doubling in mixed homogeneous and extended twisted Edwards coordinates [25]

Input: $P_1 = (X_1, Y_1, Z_1)$ in homogeneous projective coordinates.
Output: $P_3 = 2P_1 = (X_3, Y_3, T_3, Z_3)$ in extended twisted Edwards coordinates.
1: $A \leftarrow X_1^2$; $B \leftarrow Y_1^2$; $C \leftarrow 2Z_1^2$
2: $D \leftarrow -A$; $E \leftarrow (X_1 + Y_1)^2 - A - B$; $G \leftarrow D + B$
3: $F \leftarrow G - C$; $H \leftarrow D - B$; $X_3 \leftarrow E \cdot F$
4: $Y3 \leftarrow G \cdot H$; $T_3 \leftarrow E \cdot H$; $Z_3 \leftarrow F \cdot G$

Algorithm 3. Point addition in extended twisted Edwards coordinates [25]

Input: $P_1 = (X_1, Y_1, T_1, Z_1)$ and $P_2 = (X_2, Y_2, T_2, Z_2)$ in extended twisted Edwards coordinates; constant $k = -2d$, where $d = -121665/121666$.
Output: $P3 = (X_3, Y_3, T_3, Z_3)$ in extended twisted Edwards coordinates.
1: $A \leftarrow (Y_1 - X_1) \cdot (Y_2 - X_2)$; $B \leftarrow (Y_1 + X_1) \cdot (Y_2 + X_2)$; $C \leftarrow k \cdot T_1 \cdot T_2$;
2: $D \leftarrow 2Z_1 Z_2$; $E \leftarrow B - A$; $F \leftarrow D - C$;
3: $G \leftarrow D + C$; $H \leftarrow B + A$; $X_3 \leftarrow E \cdot F$;
4: $Y_3 \leftarrow G \cdot H$; $T_3 \leftarrow E \cdot H$; $Z_3 \leftarrow F \cdot G$;

Algorithm 4. Goundar's signed-digit left-to-right algorithm [24]

Input: Point $P \in E(\mathbb{F}_p)$, $k = (k_{n-1}, \ldots, k_1, k_0)_2 \in \mathbb{N}$ with $k_0 = 1$
Output: $Q = [k] \cdot P$
1: $R_0 \leftarrow P$; $R_1 \leftarrow P$
2: **for** i **from** $n-1$ **to** 1 **do**
3: $\quad t \leftarrow (-1)^{1+k_i}$
4: $\quad R_0 \leftarrow 2R_0 + (t)R_1$
5: **end for**
6: **return** R_0

Algorithm 5. Constant-time equality test (CCMP) (AVR assembly code)

Input: Registers R_i and R_t.
Output: register R_d is: 1, if $R_i = R_t$; and 0, otherwise.

```
mov  Rd, Ri
sub  Rd, Rt     ; Z (Zero) flag will be 1, if Rd == Rt; and 0, otherwise.
in   Rd, SREG   ; Rd := SREG, SREG is the status register.
andi Rd, 0x02   ; isolate Z flag.
lsr  Rd         ; Rd = Z
```

Algorithm 6. Flash table lookup protected against address leakage through power

Input: table T with dimensions n x m, where n is the number of points and m is the length in words of a point; and *index* is the index of the point. Here, T[i][j] means the value of the word and also the reading of the said word from Flash by the `LPM` instruction.
Output: m-word array r containing the words of the requested point, i.e, r = T[index][0..$m-1$].

1: **for** j **from** 0 **to** $m-1$ **do**
2: $r[j] \leftarrow 0$
3: **end for**
4: **for** i **from** 0 **to** $n-1$ **do**
5: $mask \leftarrow$ CCMP(i, index)-1 `/* mask = 0, if i = index. Otherwise, mask = 0xff */`
6: **for** j **from** 0 **to** $m-1$ **do**
7: $r[j] \leftarrow r[j] \oplus (T[i][j] \wedge mask)$
8: **end for**
9: **end for**