Faster software for fast endomorphisms

Billy Bob Brumley

Department of Pervasive Computing
Tampere University of Technology, Finland
billy.brumley@tut.fi

Abstract. GLV curves (Gallant et al.) have performance advantages over standard elliptic curves, using half the number of point doublings for scalar multiplication. Despite their introduction in 2001, implementations of the GLV method have yet to permeate widespread software libraries. Furthermore, side-channel vulnerabilities, specifically cache-timing attacks, remain unpatched in the OpenSSL code base since the first attack in 2009 (Brumley and Hakala) even still after the most recent attack in 2014 (Benger et al.). This work reports on the integration of the GLV method in OpenSSL for curves from 160 to 256 bits, as well as deploying and evaluating two side-channel defenses. Performance gains are up to 51%, and with these improvements GLV curves are now the fastest elliptic curves in OpenSSL for these bit sizes.

Keywords: elliptic curve cryptography, GLV curves, side-channel analysis, timing attacks, cache-timing attacks, OpenSSL

1 Introduction

With respect to performance, the most critical operation in an elliptic curve cryptography (ECC) implementation is scalar multiplication. The most common methods for scalar multiplication are analogous to modular exponentiation, but with signed digit sets since inverting elliptic curve points is simple. However, the number of elliptic curve point doublings in all of these methods is essentially the same, and bounded by the bit length of the scalar.

In 2001, Gallant et al. show how to halve the number of point doublings through clever choice of elliptic curve parameters [9]. These so-called "GLV curves" can exploit a fast curve endomorphism that can be computed on-the-fly and splits a single scalar multiplication into a 2-dimension multi-scalar multiplication with both arguments of roughly half the bit length.

While the theoretical gains from the GLV method are well understood, unfortunately it has yet to find its way into elliptic curve software libraries. For example, OpenSSL supports a number of GLV curves as named curves with bit sizes ranging from 160 to 256 bits but treats them as generic elliptic curves – that is, the software does not exploit the fast endomorphism for efficient scalar multiplication.

Adding to the list of deficiencies, the elliptic curve portion of OpenSSL has known side-channel vulnerabilities. In 2009, Brumley and Hakala present the first

public cache-timing attack against ECC in OpenSSL – using the vulnerability to recover a 160-bit ECC private key [5]. In 2014, Benger et al. build on that work and recover a 256-bit ECC private key with less queries [2]. The vulnerabilities remain unpatched to this day.

Motivated by these deficiencies, this work reports results of integrating the GLV method into the OpenSSL code base (Sec. 3). The performance numbers in Sec. 4 show up to a 51% improvement. Furthermore, results of integrating two side-channel defenses show that up to 33% improvement can be retained in tandem with the GLV method. Lastly, this work also evaluates said side-channel defenses to assess their effectiveness against data and instruction cache-timing attacks.

2 Background

This section contains background on GLV curves, OpenSSL's implementation of ECC and supported standard curves, and side-channel attacks on said implementation.

2.1 GLV curves

The speed at which an ECC software library performs scalar multiplication is an extremely important metric. Most of the methods are some variant of a left-to-right double-and-add algorithm, perhaps with large (signed) digit sets. While the average number of point additions will vary depending on the specific method chosen, the number of point doublings is essentially the same – the bit length of the scalar.

In 2001, Gallant et al. show that clever choice of curve parameters can actually halve the number of point doublings [9]. While the authors consider a number of different curve types, their Example 4 is the most relevant to this paper [9, Sec. 2]. Let $p = 1 \bmod 3$ and consider the following curve.

$$E(\mathbb{F}_p) : y^2 = x^3 + b$$

For this choice of p, there exists $\beta \in \mathbb{F}_p^*$ where $\text{ord}(\beta) = 3$. Observe $(x_1, y_1) \in E$ implies $(\beta x_1, y_1) \in E$. Denote this as a curve endomorphism $\phi : E \to E$ by $\phi : (x, y) \mapsto (\beta x, y)$. Denote $n = \#E$. Then $\phi(P) = \lambda P$ for some $\lambda \in \mathbb{F}_n^*$ where $\lambda^2 + \lambda + 1 = 0 \bmod n$.

While this is all very rigorous, it is not obvious how it is at all useful for scalar multiplication. The trick is to write $k = k_1 + k_2\lambda \bmod n$ for some $k_1, k_2 \approx \sqrt{n}$. Then $kP = k_1P + k_2\lambda P = k_1P + k_2\phi(P)$ and when applying ϕ on-the-fly and with k_1, k_2 half the bit length of k, this computation takes half the doublings with a 2-dimension multi-scalar multiplication method. The authors also give an algorithm to decompose scalars accordingly [9, Sec. 4]. A number of important standards feature GLV curves, details of which will be discussed later in this paper.

The number of curves for which the GLV method applies is fairly limited. More recently, Galbraith et al. show how to apply it to larger classes of curves [8]. The key idea for these GLS curves is to work over small extension fields – otherwise the scalar multiplication methods are analogous to those used in GLV. While GLS is certainly a design trend for high-speed ECC [7], unfortunately it has not seen standards adoption yet.

2.2 ECC in OpenSSL

OpenSSL first featured support for ECC in 2005. What follows is a discussion on the range of curve support in OpenSSL, some of the internal algorithms (e.g., scalar multiplication), and the side-channel weaknesses of the library.

Standardized curves. While the OpenSSL library has support for arbitrary elliptic curves in short Weierstrass form, the ones most commonly used are the so-called "named curves". For the purposes of this work, the most interesting standardized curves supported by OpenSSL are secp160r1, nistp192, nistp224, nistp256, secp160k1, secp192k1, secp224k1, and secp256k1. These are accordingly curves from NIST or SECG. The first four are curves over prime fields in short Weierstrass form with $A = -3$ and the latter four similar but with $A = 0$ and $p = 1 \bmod 3$, i.e., GLV curves. For the NIST curves OpenSSL has dedicated code for fast modular reduction – the others it uses Montgomery reduction.

Scalar multiplication. In OpenSSL, the low level scalar multiplication algorithm used depends on many factors. Each curve has an associated method structure, that contains function pointers to common ECC operations, one of these being a fully generalized multi-scalar multiplication:

```
int EC_POINTs_mul(const EC_GROUP *group, EC_POINT *r, const BIGNUM *n,
    size_t num, const EC_POINT *p[], const BIGNUM *m[], BN_CTX *ctx);
```

where `r` stores the result, `n` is the scalar to multiply the generator by, `p` is an array of `num` points, and `m` is an array of corresponding scalars to multiply said points by.

But this is just the API. The actual algorithm used varies depending on the curve by setting this function pointer when instantiating curves. For example, for all curves over binary fields the called function is an iterated implementation of Montgomery's powering ladder, in fact explicitly the algorithm given by López and Dahab [12].

For curves over prime fields and of particular interest to this work, the default method uses modified windowed Non-Adjacent Form (NAF) for scalar representation. The path through the code depends on several runtime factors. OpenSSL includes functionality for precomputing limited multiples of fixed points such as generators. The simplest case is when this precomputation is not available or not particularly helpful (e.g., no `n` given or `num` is non-zero). In this case, the algorithm is Möller's interleaved scalar multiplication [13]. The code computes

NAF for each scalar, calculates small multiples for each point depending on the digit set, then proceeds MSD to LSD doubling an accumulator point at each step, then looking at each scalar's digit in that position and adding the corresponding point to the accumulator for non-zero digits. That is, with respect to the precomputation each scalar is considered independently from others and omits linear combinations of the points across scalars. To be concrete, this is the traversed code path in the following cases:

- ECDSA signature verification.
- ECDSA signature generation if no precomputation is available.
- ECDH for unknown points.
- ECDH for fixed points if no precomputation is available.

In principle, the same code applies when precomputation is available. The precomputation strategy is to reduce the number of point doublings by computed small multiples of $2^j P$ for various values of j that allow scalar NAFs to be split into smaller chunks – Möller calls this NAF splitting [14]. For example, with secp256k1 a 256-bit scalar gets split up into roughly 32 chunks with 8 digits each. Comparing the two methods, when no precomputation is available each step performs one point doubling and looks at one or two NAF digits (depending on the number of scalars involved), whereas with precomputation each step looks at a large number of digits (e.g., 32) and in total there are only a handful of point doublings (e.g., 7). To be concrete, this is the traversed code path in the following cases:

- ECDSA signature generation when precomputation is available.
- ECDH for fixed points when precomputation is available.

Implementation attacks. OpenSSL is a popular academic target for side-channel attacks. With respect to this paper, there are two existing works that are particularly relevant.

In 2009, Brumley and Hakala show a vulnerability in OpenSSL's ECC implementation that leads to ECDSA private key recovery [5]. It is an access-driven data cache-timing attack that utilizes a local spy process that is polluting the L1 data cache in parallel (yet in a different user space) to the digital signature computation. The attack works by recovering the sequence of point doublings and additions, then filtering out very specific digital signatures for which the scalar (nonce) contains long runs of doublings, hence many consecutive zero digits in NAF representation. The authors are able to succeed in recovering an secp160r1 private key with a lattice attack after querying as few as 1K digital signatures [5, Sec. 6]. The authors describe some countermeasures [5, Sec. 7], but there is no record of source code patches on the OpenSSL development mailing list. In fact, their "Shared Context" countermeasure was later shown to be ineffective [6].

The previous attack throws away the majority of digital signatures, side-channel trace data, and potential scalar digit information. Also the spy process

targets the L1 data cache on microprocessors supporting Simultaneous Multi-Threading (SMT) through HyperThreading (HT) on Intel chips, hence does not immediately carry over to non-SMT chips. Building on this previous result, Benger et al. develop a Flush+Reload cache-timing attack in 2014 that targets the last level cache (LLC) [2]. They recover a larger private key from secp256k1 with a lattice attack after querying as few as 200 digital signatures [2, Sec. 4.1]. Furthermore, SMT or HT are not prerequisites – only a multi-core setting, hence the attack has a wider range of application. The authors describe some countermeasures [2, Sec. 5], but there is no record of source code patches on the OpenSSL development mailing list.

3 Fast and secure software

For elliptic curves that admit a fast endomorphism, it is clear that the GLV method provides significant performance gains. Furthermore, a number of GLV curves are already present in various standards – good examples are RFC4492[1] for TLS and the Bitcoin protocol specification[2]. Further still, widespread libraries like OpenSSL support these curves.

Yet the fact remains that the GLV method has yet to permeate these implementations. Filling the gap, This section describes integrating the GLV method in OpenSSL. At the same time, it addresses side-channel attacks by integrating two specific countermeasures in OpenSSL.

3.1 GLV in OpenSSL

OpenSSL treats GLV curves as "normal" curves and does not exploit their fast endomorphisms in any way. Having said that, there are at least two important use cases for the GLV method of scalar multiplication where OpenSSL could potentially benefit.

- For ECDH operations, splitting a single scalar multiplication into a 2-dimension multi-scalar multiplication with scalars of half the bit length (hence point doublings).
- For ECDSA signature verification, splitting a 2-dimension multi-scalar multiplication into a 4-dimension multi-scalar multiplication with scalars of half the bit length (hence point doublings).

In situations where OpenSSL has precomputation available at runtime, the GLV method is not useful because OpenSSL will already use interleaving, exploiting the endomorphism $P \mapsto 2^j P$. Indeed, the novelty of the GLV method lies in exploiting a *fast* endomorphism, i.e., one that is computationally efficient at runtime.

The OpenSSL code base has a few features that make implementing the GLV method quite modular and non-intrusive. The first is the fact that it already

[1] `https://tools.ietf.org/html/rfc4492`
[2] `https://en.bitcoin.it/wiki/Protocol_specification#Signatures`

contains a fully generalized multi-scalar multiplication algorithm. The second is the fact that the scalar multiplication method is controlled by a function pointer when instantiating the curve. With these observations, this work implements a new method in OpenSSL and assigns the function pointer for the GLV curves accordingly. Said function is essentially just a wrapper around the generalized multi-scalar multiplication algorithm – it decomposes each scalar into two scalars and applies the fast curve endomorphism to the corresponding points.

The above description is exactly how the implementation meets the two mentioned use cases. The exception is when precomputation exists – in that case, the code falls back to the default method since the GLV method is of no benefit. For example, in OpenSSL this might occur in ECDSA signature generation.

3.2 Regular scalar encodings

Side-channel attacks such as data and instruction cache-timing attacks can exploit implementations where the sequence of elliptic curve operations depends on the key. Ideally, as the scalar multiplication algorithm is executing it presents a consistent view through these caches that is independent of the key, i.e., the sequence of point additions and doublings is fixed regardless of the scalar. One way to do this, especially with GLV in mind, would be a multi-scalar version of Montgomery's ladder (see, e.g., Bernstein [4]) – but this would have quite a large performance penalty for OpenSSL. An ideal solution with respect to the OpenSSL code base, to retain performance and for easy integration, has the following characteristics:

- Leaves the multi-scalar multiplication algorithm largely in tact.
- Uses the same digit set as NAF.
- Does not affect the precomputation strategy.

With these goals, perhaps the most elegant solution is simply a "zero-free" scalar encoding that can serve as a drop-in replacement for OpenSSL's NAF encoding function. This work uses the "(Odd) Signed-Digit Recoding Algorithm" described by Joye and Tunstall [10, Sec. 3.2]:

> The goal is to rewrite the exponent into digits that take odd values in $\{-(2^w-1),\ldots,1,1,\ldots,2^w-1\}$

and note this digit set is exactly the same as NAF but contains no zeros. The authors accomplish this intuitively by choosing signed digits such that the remaining integer to be expanded is always odd. For brevity, this work refers to this encoding as Regular NAF (RNAF). Modifying OpenSSL's multi-scalar multiplication algorithm to utilize this encoding requires only a two line change in the function, hence is minimally invasive.

3.3 Software multiplexing

Encoding the scalar to produce a fixed sequence of point additions and doublings is enough to thwart instruction cache-timing attacks. An instruction cache-timing trace will reveal this sequence to an attacker, but it is already known a

priori. On the other hand, data cache-timing attacks are still a concern. Specifically, each point addition is a table lookup where the index is a scalar digit. A data cache-timing trace can reveal these digits to an attacker.

Software multiplexing is a tool that can be leveraged to remove these traditional table lookups. In general, the approach is useful for removing any kind of conditional branch in software, including if statements and table lookups. Software multiplexing is a well-understood method to cryptographers – two examples from the literature are particularly relevant for this work.

For Curve25519 [3] finite field arithmetic, Bernstein works in an equivalence class using a representation that is not necessarily the canonical smallest non-negative residue. This allows easier modular reductions without conditional statements – better for security, better for performance to not stall the pipeline, and better for parallelization. However, at the end of scalar multiplication the resulting point must have coordinates that are the smallest non-negative residue. Bernstein does this by subtraction and building a mask from the sign, then selecting the proper value with bitwise operations.

However, table lookups slightly differ. The best example from the literature is Käsper's work for the nistp224 elliptic curve [11, Sec. 3.4]:

> We loop through the whole precomputation table in a fixed order. While the execution time is still dependent on cache behaviour, the timing variance is independent of the secret lookup index, thus leaking no valuable timing information.

That means for each table lookup, the code traverses the entire table, and the correct value extracted from the table with bitwise operations. The mask in this case gets built from the actual lookup table index. This work uses `(index^target)-1` and a signed right shift to build the mask. It is critically important to work on the data *values* and not the *pointers*.

Integrating software multiplexing into OpenSSL's generalized scalar multiplication routine is minimally intrusive. Preceding the point addition step, the select function gets called to prepare the argument for the point addition step. Then the argument for the point addition step is the output from the select function instead of the point directly from the lookup table.

4 Results

This section looks at the performance of the code after the aforementioned modifications to OpenSSL 1.0.1l. It closes with a side-channel evaluation of the code to assess the effectiveness of the countermeasures against data and instruction cache-timing attacks.

4.1 Performance

The two ECC primitives of interest here are ECDH and ECDSA. The performance numbers compare four different bit lengths ranging from 160 to 256-bit

curves, and also compares each GLV curve with a non-GLV curve for a baseline. The benchmarking environment is an Intel Core i5-4570 (Haswell-DT, 22nm) clocked at 3.2GHz with 16GB memory running 64-bit Red Hat 6.6 "Santiago". The metric is operations per second, not clock cycles per operation. The reason for this is that OpenSSL's internal benchmarking does it that way, and that is what produced the performance numbers (specifically `openssl speed ecdh` and `openssl speed ecdsa`).

Key agreement performance. For ECDH operations, the OpenSSL `speed` utility measures the time to compute the ECDH shared secret from the private scalar and the public point. Hence it is essentially benchmarking the speed of unknown point scalar multiplication – no precomputation is available. Table 1 lists the results. The modifications to support the GLV method bring between $46 - 51\%$ performance improvement – when comparing each GLV curve to the corresponding non-GLV curve, the former is now significantly faster in all cases. The remaining columns quantify the cost of the side-channel countermeasures, both separately and in tandem. For the non-GLV curves the cost is between $8 - 9\%$. One of the most interesting observations is that with the GLV and side-channel defense modifications, the GLV curves *still* outperform the stock non-GLV curves with no such defenses.

Table 1. ECDH operations per second. The * denotes "where applicable".

curve	stock	GLV	GLV*+RNAF	GLV*+MUX	GLV*+RNAF+MUX
secp160r1	6824.1	—	6222.6	6171.9	6204.4 (-9.1%)
nistp192	5707.6	—	5317.4	5280.6	5198.8 (-8.9%)
nistp224	4077.2	—	3739.0	3785.5	3753.0 (-8.0%)
nistp256	3651.3	—	3296.1	3317.2	3319.5 (-9.1%)
secp160k1	6156.4	9292.0 (50.9%)	8173.8	8214.4	8175.9 (32.8%)
secp192k1	5181.2	7826.9 (51.1%)	6880.4	6864.7	6721.2 (29.7%)
secp224k1	3784.0	5527.7 (46.1%)	4955.4	5004.3	4891.6 (29.3%)
secp256k1	3265.8	4851.1 (48.5%)	4253.3	4276.7	4357.6 (33.4%)

Digital signature performance. For ECDSA signature generation, the OpenSSL `speed` utility utilizes precomputation. Hence the code path for the GLV method will not be exercised, and the only numbers to collect are the costs of the side-channel defenses. Table 2 holds the results. In tandem, the total cost of the side-channel defenses ranges between $15 - 20\%$. As expected, generally each GLV curve has similar performance to the corresponding non-GLV curve.

On the other hand, for ECDSA verifications OpenSSL will not use the precomputation. Also there is no need for side-channel defenses on the verification path because all the inputs are public. Table 3 holds the results. The improvements for the GLV curves, ranging from $29 - 34\%$, are due to splitting the

Table 2. ECDSA signatures generated per second.

curve	stock	RNAF	MUX	RNAF+MUX
secp160r1	19256.8	16247.6	16278.3	16244.6 (-15.6%)
nistp192	15968.0	13089.3	13099.0	13079.3 (-18.1%)
nistp224	12954.2	10259.1	10326.3	10348.5 (-20.1%)
nistp256	11067.5	8881.7	8896.7	8872.9 (-19.8%)
secp160k1	18970.1	16131.7	16118.3	16121.2 (-15.0%)
secp192k1	15629.7	12834.8	12819.6	12814.7 (-18.0%)
secp224k1	12676.9	10441.1	10383.2	10352.5 (-18.3%)
secp256k1	10760.9	8708.1	8687.7	8682.9 (-19.3%)

2-dimension multi-scalar multiplication to a 4-dimension one. When comparing each GLV curve to the corresponding non-GLV curve, the former is now significantly faster in all cases.

Table 3. ECDSA signatures verified per second.

curve	stock	GLV
secp160r1	5526.9	—
nistp192	4623.8	—
nistp224	3353.6	—
nistp256	2912.5	—
secp160k1	5131.7	6676.5 (30.1%)
secp192k1	4251.0	5484.0 (29.0%)
secp224k1	3202.3	4175.2 (30.4%)
secp256k1	2730.8	3672.9 (34.5%)

4.2 Security

The goal of this section is to provide some evidence that the side-channel defenses are effective. To this end, what follows is trace analysis for data (see [6] for the spy code) and instruction (see [1] for the spy code) cache-timing traces procured by spy processes on a microprocessor with HT. These are L1 traces for a cache with 64 sets. The spy process is executing in parallel with an OpenSSL application performing either ECDH or ECDSA signature generation for curve secp256k1. The unprotected version in the ECDH case is inclusive of the GLV method, but with no side-channel defenses. As previously discussed, GLV method does not apply to the code path for ECDSA signature generation.

ECDH analysis. Figure 1 shows the instruction cache traces. With no defenses (top), the red annotation shows a number of point doublings while the blue annotation shows two point additions separated by a point doubling – this leak reveals key material. With defenses (bottom), the red annotation shows two

consecutive point additions while the blue annotation shows a number of point doublings. This sequence repeats throughout the trace, showing the effectiveness of the RNAF defense.

Figure 2 shows the data cache traces. With no defenses (top), the red annotation shows two point additions separated by a point doubling. The blue annotation shows two consecutive point additions. These leaks reveal key material. The trace shows the digits used in the lookup table are different because of the varying latency in many of the cache sets. With defenses (bottom), the trace is quite different. Two point additions, annotated in red, clobber a large number of cache sets. This is then followed by a number of point doublings in blue. What this suggests is the effectiveness of the MUX defense since the code traverses the entire table and has a big footprint on the cache, and furthermore the effectiveness of the RNAF defense since this sequence is essentially repeated throughout the trace.

ECDSA analysis. Figure 3 shows the instruction cache traces. With no defenses (top), the red annotation shows all 7 point doublings – with the interleaving method in this case there are 32 chunks with 8 digits each, so exactly 7 point doublings occur. Between these, some annotated in blue, are a varying number of point additions – this leak reveals key material. With defenses (bottom), the red annotation shows four of the 7 consecutive point doublings: RNAF fixes the sequence, so the other 3 doublings appear consecutively towards the beginning of the trace (not shown). while the blue annotation shows a number of point doublings. Everything that remains is point additions. This sequence of operations reveals nothing to the attacker – the RNAF defense is working as expected.

Figure 4 shows the data cache traces. With no defenses (top), the red annotation shows one of the point doubling while the blue annotation shows a number of consecutive point additions. Observe the number of point additions between doublings varies and the latency of many of the lower cache sets reveals distinct digits used in the lookup table – this leak directly reveals key material. With defenses (bottom), the red annotation simply shows a number of point additions and highlights the fact that essentially all cache sets get clobbered as a result of the MUX defense – it is working as intended.

5 Conclusion

Using OpenSSL as a case study[3], the goal of this work is to give concrete numbers on the performance improvement realized with the GLV method, as well as to address the known side-channel vulnerabilities in OpenSSL ECC. To that end, the contributions of this work are as follows:

– Up to 51% performance improvement for GLV curves without side-channel defenses.

[3] `http://rt.openssl.org/Ticket/Display.html?id=3667&user=guest&pass=guest`

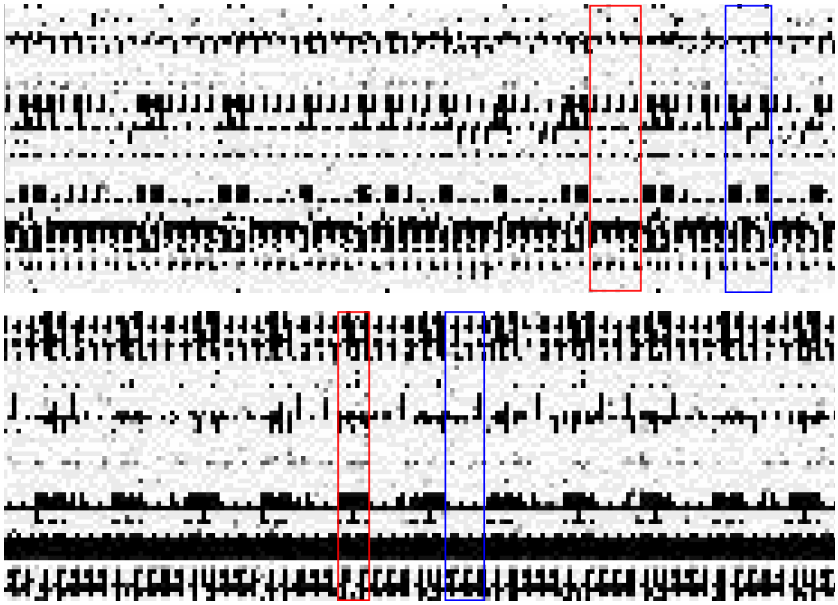

Fig. 1. ECDH through the instruction cache: with (bottom) and without (top) side-channel mitigations. y-axis: cache set index. x-axis: time. Gradient: latency from black (low) to white (high).

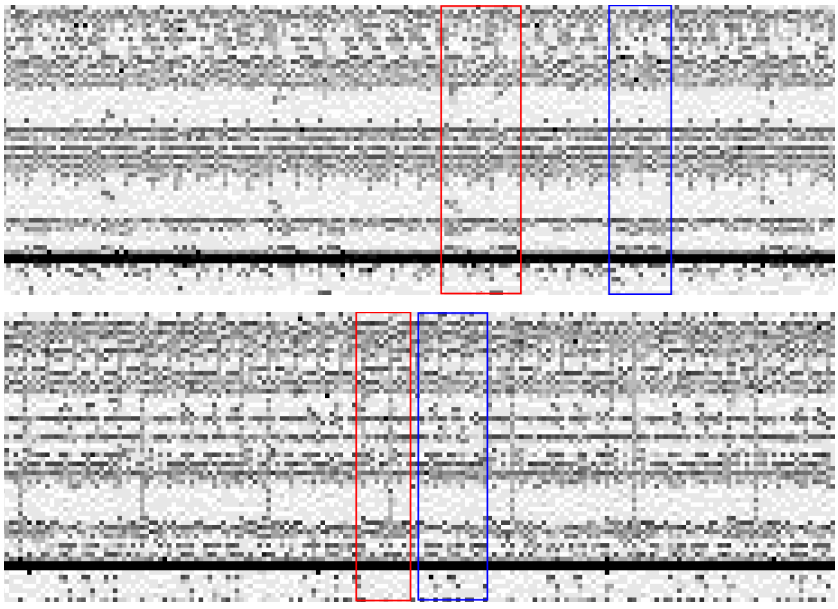

Fig. 2. ECDH through the data cache: with (bottom) and without (top) side-channel mitigations. y-axis: cache set index. x-axis: time. Gradient: latency from white (low) to black (high).

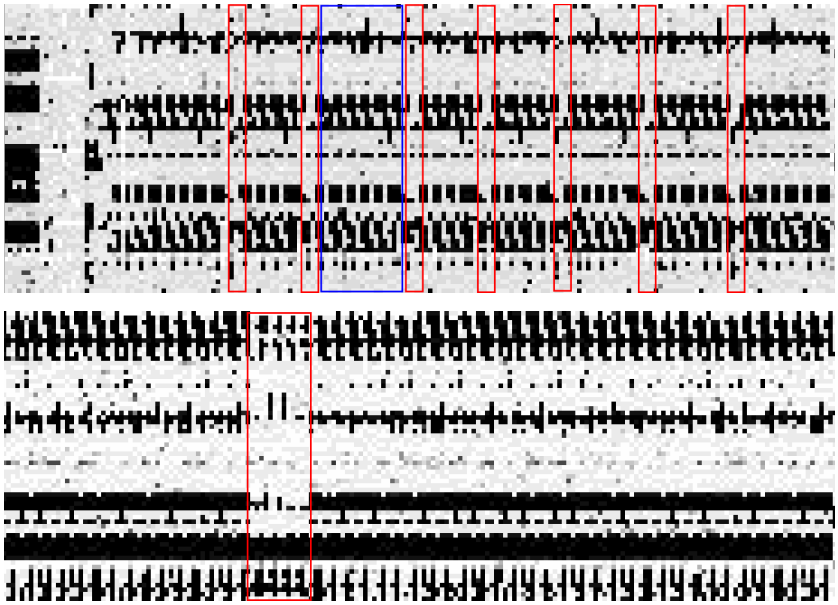

Fig. 3. ECDSA through the instruction cache: with (bottom) and without (top) side-channel mitigations. y-axis: cache set index. x-axis: time. Gradient: latency from black (low) to white (high).

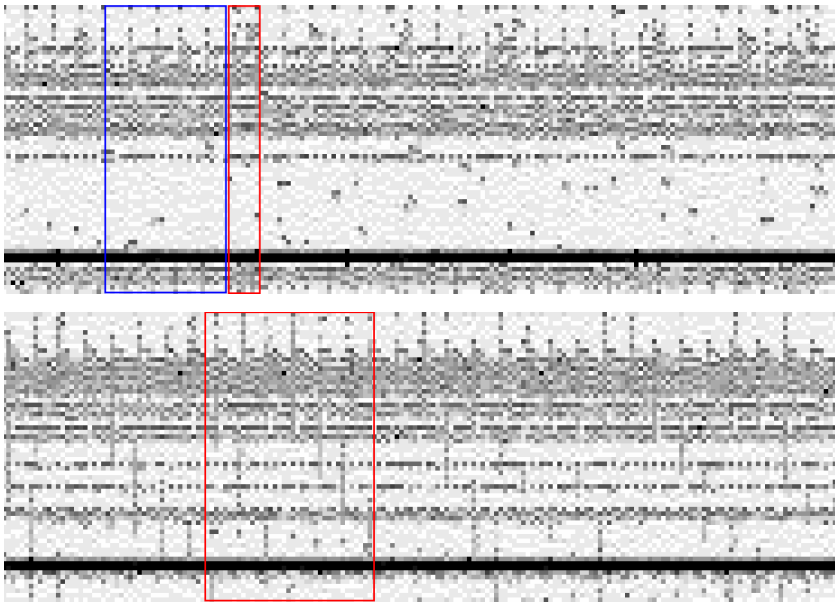

Fig. 4. ECDSA through the data cache: with (bottom) and without (top) side-channel mitigations. y-axis: cache set index. x-axis: time. Gradient: latency from white (low) to black (high).

- Up to 33% performance improvement for GLV curves with side-channel defenses.
- GLV curves *with* side-channel defenses now outperform non-GLV curves *without* side-channel defenses.
- First concrete solution (i.e., source code patch) for OpenSSL's known ECC side-channel vulnerabilities.
- Concrete evaluation of ECC software side-channel defenses, in contrast to other works that rather design for side-channel security without a platform evaluation.
- Within OpenSSL, the first application of multi-scalar multiplication for more than two scalars – better utilizing the generalized multi-scalar multiplication algorithm already present in the library.

In conclusion, this work shows that fast and secure ECC is possible for a widely-deployed software library – the concepts are not mutually exclusive.

One last subtle observation resulting from this work is that the side-channel methods to attack ECDSA depend heavily on the target application. Both published attacks [5,2] target only applications where ECDSA precomputation is *not* available – most likely not a conscience choice by the authors, but a practical difference nonetheless. This work highlights that the methods to attack applications *with* precomputation are likely very different than those without – neither of the previous attacks observe this nuance.

References

1. Acıiçmez, O., Brumley, B.B., Grabher, P.: New results on instruction cache attacks. In: Mangard, S., Standaert, F. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6225, pp. 110–124. Springer (2010), http://dx.doi.org/10.1007/978-3-642-15031-9_8
2. Benger, N., van de Pol, J., Smart, N.P., Yarom, Y.: "Ooh aah... just a little bit" : A small amount of side channel can go a long way. In: Batina, L., Robshaw, M. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8731, pp. 75–92. Springer (2014), http://dx.doi.org/10.1007/978-3-662-44709-3_5
3. Bernstein, D.J.: Curve25519: New Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings. Lecture Notes in Computer Science, vol. 3958, pp. 207–228. Springer (2006), http://dx.doi.org/10.1007/11745853_14
4. Bernstein, D.J.: Differential addition chains (2006), http://cr.yp.to/ecdh/diffchain-20060219.pdf
5. Brumley, B.B., Hakala, R.M.: Cache-timing template attacks. In: Matsui, M. (ed.) Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan,

December 6-10, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5912, pp. 667–684. Springer (2009), http://dx.doi.org/10.1007/978-3-642-10366-7_39

6. Brumley, B.B., Tuveri, N.: Cache-timing attacks and shared contexts. In: Constructive Side-Channel Analysis and Secure Design - 2nd International Workshop, COSADE 2011, Darmstadt, Germany, February 24-25, 2011. Proceedings. pp. 233–242 (2011)
7. Faz-Hernández, A., Longa, P., Sánchez, A.H.: Efficient and secure algorithms for GLV-based scalar multiplication and their implementation on GLV-GLS curves (extended version). J. Cryptographic Engineering (2014), http://dx.doi.org/10.1007/s13389-014-0085-7
8. Galbraith, S.D., Lin, X., Scott, M.: Endomorphisms for faster elliptic curve cryptography on a large class of curves. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5479, pp. 518–535. Springer (2009), http://dx.doi.org/10.1007/978-3-642-01001-9_30
9. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: Kilian, J. (ed.) Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2139, pp. 190–200. Springer (2001), http://dx.doi.org/10.1007/3-540-44647-8_11
10. Joye, M., Tunstall, M.: Exponent recoding and regular exponentiation algorithms. In: Preneel, B. (ed.) Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5580, pp. 334–349. Springer (2009), http://dx.doi.org/10.1007/978-3-642-02384-2_21
11. Käsper, E.: Fast elliptic curve cryptography in OpenSSL. In: Danezis, G., Dietrich, S., Sako, K. (eds.) Financial Cryptography and Data Security - FC 2011 Workshops, RLCPS and WECSR 2011, Rodney Bay, St. Lucia, February 28 - March 4, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7126, pp. 27–39. Springer (2011), http://dx.doi.org/10.1007/978-3-642-29889-9_4
12. López, J., Dahab, R.: Fast multiplication on elliptic curves over GF(2^{m}) without precomputation. In: Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1717, pp. 316–327. Springer (1999), http://dx.doi.org/10.1007/3-540-48059-5_27
13. Möller, B.: Algorithms for multi-exponentiation. In: Vaudenay, S., Youssef, A.M. (eds.) Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16-17, 2001, Revised Papers. Lecture Notes in Computer Science, vol. 2259, pp. 165–180. Springer (2001), http://dx.doi.org/10.1007/3-540-45537-X_13
14. Möller, B.: Improved techniques for fast exponentiation. In: Lee, P.J., Lim, C.H. (eds.) Information Security and Cryptology - ICISC 2002, 5th International Conference Seoul, Korea, November 28-29, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2587, pp. 298–312. Springer (2002), http://dx.doi.org/10.1007/3-540-36552-4_21