Applying TVLA to Public Key Cryptographic Algorithms

Michael Tunstall and Gilbert Goodwill

Abstract

Test Vector Leakage Assessment (TVLA) has been proposed as a method of determining if a side-channel attack is feasible, for a given implementation of a block cipher, by looking for leakage without conducting an attack. The thresholds chosen for the evaluation of leakage are chosen such that passing the tests gives a strong indication that no leakage is present. In this document, we describe how TVLA can be adapted to pubic key cryptographic algorithms, with a specific focus on RSA, ECDSA and ECDH.

1 Introduction

Side-channel analysis was introduced to the cryptographic community by Kocher who noted that the time required to compute a modular exponentiation could potentially reveal the exponent that was used [1]. Further work by Kocher et al. demonstrated the instantaneous power consumption was dependent on the data being manipulated at a given point in time [2]. This would allow cryptographic keys to be determined by making hypotheses of intermediate states of a cryptographic algorithm and validating these hypotheses using the instantaneous power consumption. The same analysis has also been shown to be possible on the electromagnetic emanations of a microprocessor, where acquisitions are treated in the same way [3, 4]. In the following, we shall refer to power consumption acquisitions but the same treatment can be applied to acquisitions of the electromagnetic emanations around a microprocessor.

Implementations of cryptographic algorithms need to be tested on certain devices to ensure that they are not vulnerable to such attacks. Initially, this involved testing whether the effect of specific intermediates could be observed using a test based on a difference-of-means test [2] or Pearson's correlation coefficient [5]. However, this only assures that an attack on a specific intermediate state is not possible and other intermediate states may still leak.

A more general approach for determining if the power consumption of a device relates to the data it is manipulating, referred to as Test Vector Leakage Assessment (TVLA), has been proposed by Goodwill et al. [6] with further detail provided by Schneider and Moradi [7]. TVLA uses a t-test to evaluate

the differences between sets of acquisitions to determine if one set can be distinguished from the other. This provides a robust test that is simultaneously applied to numerous intermediates, providing a clear indication of leakage or lack thereof.

In this document, we present a strategy for applying TVLA to public-key algorithms, specifically focusing on RSA, ECDSA and ECDH as the most commonly used algorithms. We focus on determining if the leakage required to conduct an attack is present, and provide details on how these tests can be conducted, given that there are many different ways of implementing these algorithms.

2 Background

One of the tests in TVLA is to determine whether there are statistically significant differences in the mean traces of two sets of traces, one acquired with a fixed plaintext and the other with random plaintexts. One would typically randomly interleave acquisitions so that environmental effects are the same for both sets and there are no erroneous indications of leakage, caused, for example, by the least significant bit of a variable used to count the number of acquisitions. In applying this, one would take two sets of data, and conduct Welch's t-test point-by-point to determine whether there is evidence against the null hypothesis that the sets are the same.

Consider two sets of acquisitions, of n_1 and n_2 samples, respectively. We can compute their respective sample means, $\bar{x}_1$ and $\bar{x}_2$, and respective sample standard deviations, σ_1 and σ_2. One can then compute a t-statistic using Welch's t-test

$$\alpha = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}},$$

where the result is distributed over a t-distribution with ν degrees of freedom (i.e., $\alpha \sim t(\nu)$). In practice, one would use the asymptotic result that the t-distribution is equivalent to the standard normal distribution, so ν does not need to be defined.

Goodwill et al. propose that observing $\alpha > 4.5$ indicates the presence of leakage. Specifically, an $\alpha > 4.5$ gives the probability of indicating leakage where no leakage is present, often referred to as a Type I error or false positive, of approximately 1×10^5. However, repeating an experiment can mitigate this problem. The probability of no leakage being indicated where leakage is present, often referred to as a Type II error or false negative, is not defined but, again, repeating an experiment can mitigate this problem. To detect side-channel leakage that could be used to conduct an attack, one would typically use this test on a set of power consumption traces where the t-statistic is computed in a point-wise manner. That is, one can test for leakage at each point in time using two sets of acquisitions. This can be extended to a second-order analysis (where we have $\alpha > 5$ indicating leakage) in the same way one would for any other side channel analysis [8].

To avoid false positive caused by plaintexts being manipulated before being masked, one can specifically target the middle of a cryptographic operation, allowing inputs and outputs that are indistinguishable from random values via some side-channel. For example, by generating random-seeming inputs to an algorithm that produce a fixed value in an algorithm at a chosen point, similar to the approach described by Mizaki and Hayashi [9]. Rather than having a test where we are comparing a fixed input with a random input, we have an input that we refer to as "semi-fixed" as it produces a fixed value at a specific point in time. The resulting set of inputs can be randomly interleaved with random inputs to provide a robust method of detecting leakage using the methods described by Goodwill et al. We note that a specific cryptographic key must accompany the plaintexts, so such a test requires the ability to set the cryptographic keys in the targeted device.

If one reveals a leak using semi-fixed inputs, the results do not specifically reveal a particular type of leak or indicate how to exploit the leakage to determine a secret key. However, by using the same technique, one can generate more specific plaintexts, with associated secret keys, to help isolate where, and how, a leakage manifests. The interested reader is referred to Schneider and Moradi [7] for a thorough treatment of this topic and leakage detection using the t-test.

3 Outline of Evaluation Strategy

The following details the general process that we follow when evaluating an implementation of a public-key cryptographic algorithm. The tests to be conducted for specific algorithms are detailed in subsequent sections. The steps of the attack are as follows:

3.1 Theoretical Analysis

The first step of the attack is to conduct a theoretical analysis of the implementation to determine which specific attacks need to be evaluated. That is, look for specific weaknesses that could be exploited by providing specific inputs to an algorithms or making hypotheses based on some particular state of a register/variable. Further detail is given below for the particular algorithms.

3.2 Timing Analysis

As a general principle, an implementation of a public-key cryptographic algorithm should conduct the same operations in the same order for all possible inputs. However, in some cases it may be necessary to break this rule. For example, using the extended GCD algorithm to compute a modular inverse, which is significantly faster than alternative algorithms.

To test if operations that will take a variable amount of time can be identified they should be timed for 1×10^6 operations where we interleave a fixed input

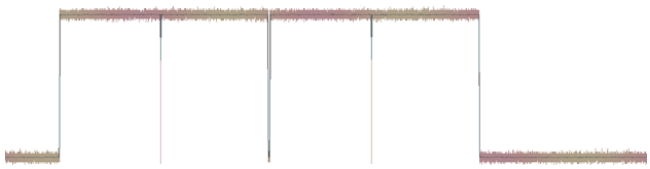

Figure 1: A sample trace showing the voltage applied to GPIO pins isolating four individual operations.

with a random input and conduct a t-test, as described above. To determine that there is no leakage there should be no observed t-statistic greater than 4.5.

One straightforward method of measuring the time required to compute a given operation is to use GPIO pins to generate a trace, as shown in Figure 1, and measuring the time between the events on an I/O pin *a posteriori*. If GPIO pins cannot be affected then a power consumption trace can be acquired and the operation measured by looking for patterns in the power consumption. The methods one would use to conduct these measurements are beyond the scope of this document. However, one would require that the measurement be cycle accurate to give a clear indication of whether an attack is feasible.

3.3 Simple Power Analysis

A side-channel resistant implementation of a cryptographic algorithm should be straight-line code, i.e., the same sequence of opcodes should be executed irrespective of the input or cryptographic key. If this is not the case one should not be able to observe any secret information by observing a small number of power consumption traces. It is advantageous to evaluate some specific functions, e.g., the conditional subtraction in Montgomery multiplication, but is not obligatory as leakage detection will reveal a SPA leak very quickly.

3.4 Leakage Detection

Leakage detection, as described above, is used to determine if sets of traces can be distinguished and, hence, if an attacker could attempt an attack based on distinguishing sets of traces. These tests are typically conducted to determine if a fixed parameter can be distinguished from a random parameter. Typically, this results in two tests:

1. A fixed message versus a random message with a fixed key, and
2. A fixed cryptographic key versus a random cryptographic key with a fixed or random message.

In all cases we take a set of randomly interleaved acquisitions from the fixed and random set, and conduct the t-test as described above, where the number

of traces taken indicates the security level required. To determine that there is no leakage there should be no observed t-statistic greater than 4.5.

Ideally, one would acquire traces that capture the power consumption during the entire computation required for a cryptographic algorithm. In practice, this may not be possible, since the required sampling frequency may produce acquisitions that are prohibitively large. In some cases, we therefore take two sets of traces showing the beginning and the end of the algorithm, where the missing part is known to be the repeated loops of an exponentiation algorithm (for example). That is, we assume that if the loops of an exponentiation algorithm do not leak at the beginning and the end of the operation, then the middle loops will be equally secure.

In practice, further tests may be required. These are listed below for each algorithm we describe in this document.

3.5 Collision Attacks

A collision attack in our context is where an attacker seeks to determine whether the input or output of one operation is used as the input to another operation in a group exponentiation algorithm. The way that a collision attack functions will depend on the exponentiation algorithm that has been chosen and will requires some effort to determine where group operations are represented in power consumption traces.

To conduct an evaluation one can, for example, choose the group operations of the third round of an exponentiation algorithm, and compare them with the group operations of the fourth round. Rather than try to generate specific test vectors, we acquire 1×10^4, or a sufficiently large number to conduct the analysis below, power consumption/EM acquisitions where the exponent used is randomly generated. Assuming a one-bit treatment of the exponent, a typical evaluation would proceed as follows:

1. For a fixed bit value for the third bit and fourth bit of the exponent, say 0, in both cases, we take 1×10^3 traces, taken from the 1×10^4 power consumption/EM acquisitions referred to above, where this is the case and extract the power consumption/EM subtraces corresponding to the operations in the third and fourth round. That is, for some trace O we split the trace up into subtraces

   $$O = \{o_1, o_2, o_3, ...o_n\}$$

   where o_i, for $1 \leq i \leq n$, corresponds to the i-th subtrace taken while the i-th operation was being computed. For example, in an exponentiation algorithm that treats an exponent bit-by-bit there will typically be two operations per round of the algorithm.

2. Generate a mean subtrace $\bar{o}$ by extracting the subtraces from all the acquisitions and generating a mean trace.

3. Subtract the mean trace $\bar{o}$ point-by-point from each subtrace giving

$$\begin{aligned}\hat{O} &= \{o_1 - \bar{o}, o_2 - \bar{o}, o_3 - \bar{o}, ..., o_n - \bar{o}\} \\ &= \{\hat{o}_1, \hat{o}_2, \hat{o}_3, ..., \hat{o}_n\},\end{aligned}$$

   where we define $\hat{o}_i = o_i - \bar{o}$ for $1 \leq i \leq n$.

4. For each trace, we take each subtrace representing each operation in the third round, compute the pointwise difference with the subtraces representing each operation in the fourth round, producing difference traces. If, for example, we assume the Montgomery Ladder [10, 11] is being used there are two operations used per round, so $\{o_5, o_6\}$ represent the third round and $\{o_7, o_8\}$ represent the fourth round.

5. Concatenate the difference traces produced from each trace to create a combined difference of operation trace. Giving, in our example, a trace

$$\Delta = \{\hat{o}_5 - \hat{o}_7, \hat{o}_5 - \hat{o}_8, \hat{o}_6 - \hat{o}_7, \hat{o}_6 - \hat{o}_8\},$$

   where the subtraction occurs point-by-point.

6. Steps 2–5 are repeated with different traces to produce 1×10^3 difference traces.

7. Repeat the process with randomly selected traces, resulting in a second set of 1×10^3 difference traces.

8. Apply the t-test, as described above, to the two sets of difference traces. That is, try to find evidence against the null hypothesis that the two sets are the same.

As previously, to determine that there is no leakage, there should be no observed t-statistic greater than 4.5. This test should be repeated with all the possible combinations of the third and fourth bits of the exponent. That is, the bits from the set $\{00, 01, 10, 11\}$.

Note that the algorithm outlined above assumes that an exponent is read bit-by-bit, such as used in the Montgomery powering ladder [10, 11] or the binary exponentiation algorithm. Some further modifications may be required for other algorithms. For example, if we consider the 4-ary exponentiation algorithm [12], each loop of the exponentiation algorithm will treat two bits of the exponent and use a look-up table based on the value of these two bits. The above would then need to be modified to determine if the multiplications used for two identical sets of two bits are distinguishable from the multiplications used for random sets of two bits.

In this test, we only consider the input of one operation being used in the input of another operation. It has been shown in the literature that it is possible to determine where the output of one operation has been used as the input to another operation [13]. However, only one example of this technique exists on a target that provides very clean acquisitions. It has not yet been shown that such an attack is possible on any device with even a moderate amount of noise in the acquisitions.

4 RSA

In this section we shall consider RSA computed as a single exponentiation that does not use the Chinese remainder theorem. In the following, we assume that the computation is the modular exponentiation of a message m, raised to the power of the private exponent d modulo n, where $\{e = 2^{16} + 1, n\}$ is an RSA public key and $\{d\}$ is an RSA private key. Hence we are evaluating the security of the operation:

$$s = m^d \mod n$$

We also consider the RSA computed using the Chinese remainder theorem, where the private key is $\{p, q, d \mod p - 1, d \mod q - 1, q^{-1} \mod p\}$ where $n = p\,q$.

The tests required to evaluate an implementation of this operation are detailed below. We shall assume that n is a k-bit integer, and p and q are $k/2$-bit integers.

4.1 Theoretical Analysis

The choice of algorithm used to implement RSA can have a large impact on the security of the resulting implementation [14]. As part of the evaluation, one determines if any of the following values could have an impact on the side-channel resistance of the implementation. That is, whether could they provoke any behavior different to a randomly generated input.

1. 0
2. 1
3. 2
4. $n - 1$
5. $n - 2$
6. $2^{-k} \mod n$ (i.e. Montgomery form equal to 1 modulo n)
7. $2^{-k+1} \mod n$ (i.e. Montgomery form equal to 2 modulo n)
8. $2^{-\frac{k}{2}} \mod n$ (i.e. square of input has Montgomery form equal to 1 modulo n)
9. input whose Montgomery form is $2^{k-1} - 1 \mod n$
10. input congruent $1 \mod p$
11. input congruent $2 \mod p$
12. $2^{-\frac{k}{2}} \mod p$ (i.e. Montgomery form equal to 1 modulo p)
13. $2^{-\frac{k}{2}+1} \mod p$ (i.e. Montgomery form equal to 2 modulo p)
14. $2^{-\frac{k}{4}} \mod p$ (i.e. square of input has Montgomery form equal to 1 modulo p)
15. input whose Montgomery form is $2^{\frac{k}{2}-1} - 1 \mod p$
16. input congruent $1 \mod q$
17. input congruent $2 \mod q$

18. $2^{-\frac{k}{2}} \mod q$ (i.e. Montgomery form equal to 1 modulo q)
19. $2^{-\frac{k}{2}+1} \mod q$ (i.e. Montgomery form equal to 2 modulo q)
20. $2^{-\frac{k}{4}} \mod q$ (i.e. square of input has Montgomery form equal to 1 modulo q)
21. input whose Montgomery form is $2^{\frac{k}{2}-1} - 1 \mod q$

Ideally, one would test all of these inputs to determine how they affect the behavior of a side-channel. This may not be possible because of the time required to take numerous sets of acquisitions and is best done by analyzing the algorithms implemented. The above list is not exhaustive and care needs to be taken to look for other special cases that may arise in a specific implementation.

One example of the effect of a specifically chosen input is shown in Figure 2, which shows the power consumption of a device during the generation of an RSA signature with two different inputs. The upper traces shows the power consumption where the input has a bit length similar to the modulus n. In the lower trace, the input is set to 2, which has a significant impact on the power consumption.

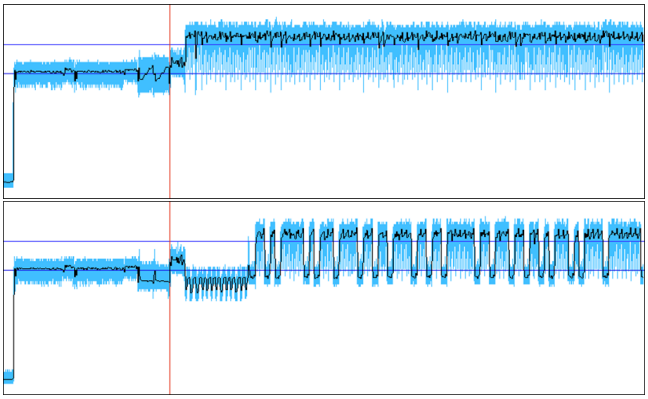

Figure 2: Two power consumption traces taken during the computation of an RSA signature generation. The upper trace is with an input with the same bit length as the exponent. The lower with the input set to 2.

4.2 Leakage Detection

An evaluation of the side-channel information available to an attacker is conducted by determining whether a fixed input or private key can be distinguished from a random input or private key, respectively.

The first test is to generate random private keys and compare the acquisitions taken with those taken using a fixed private key, using the method described in Section 3.4. No t-statistic greater than 4.5 should be visible treating traces representing the beginning and end of the RSA operation. It is assumed that each key is blinded and stored as it is loaded. Otherwise, one will see leakage where the key blinding takes place.

We then treat the input in the same way. However, we wish to determine if there is any leakage without getting false positives from the input directly affecting the side-channel. We therefore generate inputs that will produce a fixed state after a certain number of bits of the private exponent have been treated. Acquisitions are taken and compared to those taken for random inputs, using the method described in Section 2. To generate these inputs we generate an RSA key, and arbitrarily choose some random value $g < n$ which will be the state we will consider. We can then generate pairs of inputs and private key exponents that will produce an intermediate state g after a certain number of, say 16, bits of the private exponent have been treated using Algorithm 1. To conduct a test where we wish to compare a fixed input with a random input we instead compare a fixed state with a random state, for which we generate a pairs of inputs and private exponents for each case. Note that the bits of the exponent after the chosen state g is produced will be fixed so, for a fixed g, all subsequent states should also be fixed in a naïve implementation.

Algorithm 1: Generating input/key pair for RSA leakage detection

Input: An RSA key pair $\{e, n\}$ and $\{d\}$, state g, b the bit length of d, bit length of random exponent bits ℓ.

Output: m, d'

$r \leftarrow 2$;
while $\gcd(r, \phi(n)) \neq 1$ **do**
  $r \xleftarrow{R} \{1, 3, 5, ..., 2^{\ell} - 1\}$;
end
$r \leftarrow r + 2^{\ell}$;
$d' \leftarrow \left(d \wedge \left(2^{b-ell-1} - 1\right)\right) + 2^{b-\ell-1} r$;
$k \leftarrow r^{-1} \bmod \phi(n)$;
$m \leftarrow g^k \bmod n$;
return m, d'

If we consider RSA computed using the CRT, we can conduct a similar analysis but we are able to generate pairs of inputs and private keys where the analysis described above can be applied to both of the required exponentiations. That is, we can generate inputs and private keys that will provide a fixed state after each exponentiation has treated a given number of bits of each exponent. The algorithms for computing these inputs is provided in Algorithm 2. One can then conduct a leakage detection test by comparing traces acquired where the internal state is fixed compared to an input that will produce a random state.

Both Algorithm 1 and Algorithms 2 assume that the exponent is being treated left-to-right (i.e., most-significant bit to least-significant bit). If one wishes to test a right-to-left implementation, it is straightforward to modify these algorithms to insert random values in the least significant bits of exponents.

We note that the inputs generated to conduct these tests will not generate a verifiable signature, which may be problematic if the implementation being analyzed contains countermeasures to prevent fault attacks. If countermeasures to fault attacks are present, a strategy for conducting an analysis will need to be implemented. For example, a device may need to be reset if it mutes itself after an attack is detected.

Algorithm 2: Generating input/key pair for RSA-CRT leakage detection

Input: An RSA key pair $\{e, n\}$ and $\{p, q, d_p, d_q, i_q\}$, state g, b_p and b_q the bit length of d_p and d_q respectively, bit length of random exponent bits ℓ.

Output: m, $\{p, q, d'_p, d'_q, i_q\}$

$r_p \leftarrow 2$;
while $\gcd(r_p, \phi(p)) \neq 1$ **do**
  $r_p \xleftarrow{R} \{1, 3, 5, ..., 2^\ell - 1\}$;
end
$r_p \leftarrow r_p + 2^\ell$;
$r_q \leftarrow 2$;
while $\gcd(r_q, \phi(q)) \neq 1$ **do**
  $r_q \xleftarrow{R} \{1, 3, 5, ..., 2^\ell - 1\}$;
end
$r_q \leftarrow r_q + 2^\ell$;
$d'_p \leftarrow \left(d_p \wedge \left(2^{b_p - ell - 1} - 1\right)\right) + 2^{b_p - \ell - 1} r_p$;
$d'_q \leftarrow \left(d_q \wedge \left(2^{b_q - ell - 1} - 1\right)\right) + 2^{b_q - \ell - 1} r_p$;
$k_p \leftarrow r_p^{-1} \bmod \phi(p)$;
$k_q \leftarrow r_q^{-1} \bmod \phi(q)$;
$m_p \leftarrow g^{k_p} \bmod p$;
$m_q \leftarrow g^{k_q} \bmod q$;
$m \leftarrow \mathrm{CRT}(m_p \bmod p, m_q \bmod q)$;
return $m, \{p, q, d'_p, d'_q, i_q\}$

4.3 Collision Attacks

A collision attack applied to the implementation of exponentiation algorithms used in RSA can be applied as described in Section 3.5.

5 Elliptic Curve-based Algorithms

In this section, we shall consider elliptic curve-based algorithms, which use the elliptic curve $\mathcal{E}$ over finite field $\mathbb{F}_q$ consists of points (x, y), with x, y in $\mathbb{F}_q$, that satisfy the short Weierstraß equation

$$\mathcal{E} : y^2 = x^3 + a\,x + b$$

with $a = -3$ and $b \in \mathbb{F}_q$, and the point at infinity denoted $\boldsymbol{O}$. The set $\mathcal{E}(\mathbb{F}_q)$ is defined as

$$\mathcal{E}(\mathbb{F}_q) = \{(x, y) \in \mathcal{E} \,|\, x, y \in \mathbb{F}_q\} \cup \{\boldsymbol{O}\}\,,$$

where $\mathcal{E}(\mathbb{F}_q)$ forms an Abelian group under the chord-and-tangent rule and $\boldsymbol{O}$ is the identity element. The scalar multiplication of a given point is a group exponentiation in $\mathcal{E}$ that uses elliptic curve arithmetic, i.e. $[k]\,\boldsymbol{P}$ for some integer $k < |\mathcal{E}|$.

In an implementation of ECDSA, we take base point $\boldsymbol{P} = (x, y)$ for an elliptic curve $\mathcal{E}$ over $\mathbb{F}_p$, with private key d and a hash function h. To sign a message m, the

signer picks a random $k < |\mathcal{E}|$ and computes

$$r \xleftarrow{x} [k]\boldsymbol{P} \quad \text{and} \quad s \leftarrow k^{-1}\left(h(m) + d\,r\right) \bmod |\mathcal{E}|\,.$$

We denote the extraction of the x-coordinate of a point and its assignment to a variable by $\xleftarrow{x}$. The signature of m is the pair: $\{r, s\}$. We note that the security of this signature scheme relies on the random value k remaining unknown to an attacker.

In an implementation of ECDH, which also uses an elliptic curve $\mathcal{E}$ over finite field $\mathbb{F}_q$ consists of points (x, y), with x, y in $\mathbb{F}_q$, that satisfy a short Weierstraß equation.

If we consider two people with public keys (curve points) $\boldsymbol{Q_1}$ and $\boldsymbol{Q_2}$ with corresponding private keys d_1 and d_2 one party computes

$$\boldsymbol{R} = [d_1]\boldsymbol{Q_2}$$

and the other computes

$$\boldsymbol{R} = [d_2]\boldsymbol{Q_1}$$

where the point $\boldsymbol{R}$ is the same in both cases. While these keys are often ephemeral one needs to evaluate the case, where private keys are fixed for some use cases.

In evaluating implementations of ECDSA and ECDH the tests are somewhat similar. We highlight the differences below.

5.1 Theoretical Analysis

A theoretical analysis of the elliptic curve-based algorithm will look at how the choice of algorithms affects the side-channel resistance of the implementation. Specifically,

1. The algorithm used to invert the nonce, i.e., compute k^{-1}. If, for example, the extended GCD is used, a timing analysis will need to be conducted to ensure that the time required to compute the extended GCD is not dependent on the nonce k.
2. The group exponentiation algorithm used in ECDSA or ECDGH should, as much as possible, be determined to enable further testing. In particular, it is important to ensure that the algorithm is highly regular, i.e., the sequence of operations is not dependent on the nonce k (ECDSA) or the private key (ECDH).

5.2 Leakage Detection

An evaluation of the side-channel information available to an attacker is conducted by determining whether a known input or private key can be distinguished from a random input or private key, respectively.

To test the implementation of a group exponentiation algorithm the strategy is different depending on the algorithm.

5.2.1 ECDSA

The first test is to generate random private keys and compare the acquisitions taken with those taken using a fixed private key, using the method described in Section 3.4. To conduct this test one can take acquisitions during the signature generation at the end of the ECDSA computation, and the nonce can be random in all cases. No t-statistic greater that 4.5 should be visible treating traces representing the beginning

and end of the ECDSA operation. It is assumed that each key is blinded and stored as it is loaded. Otherwise, one will see leakage where the key blinding takes place.

We can then treat the other known input $h(m)$ in the same way. However, this values will be unblinded and, if there is any visible leakage, we may have false positives. We therefore generate inputs that will produce a fixed state after a given operation. We note that to conduct these tests one does need to be able to fix, or predict, the nonces that will be used during the acquisitions.

To conduct a test where we wish to compare a fixed state with a random state, for which we generate a pairs of nonces and inputs (we assume the private key is fixed). We generate inputs for the fixed state by generating some arbitrary value g and, for each known nonce, we compute a value for $h(m)$ that will give us g. One can then conduct a leakage detection test by comparing traces acquired where the internal state is fixed compared to an input that will produce a random state.

Algorithm 3: Generating input/nonce pairs for ECDSA leakage detection

Input: An ECDSA private key d, point $\boldsymbol{P} \in \mathcal{E}$ state g and the nonce k.
Output: $h(m)$ paired with k.

1 $r \xleftarrow{x} [k]\boldsymbol{P}$;
2 $m \leftarrow g - d\,r \bmod |\mathcal{E}|$;
3 **return** m

5.2.2 ECDH

The only test we need to conduct on the ECDH primitive is to generate random private keys and compare the acquisitions taken with those taken using a fixed private key, using the method described in Section 3.4. No t-statistic greater than 4.5 should be visible treating traces representing the beginning and end of the ECDH operation. It is assumed that each key is blinded and stored as it is loaded. Otherwise, one will see leakage where the key blinding takes place.

5.3 Collision Attacks

A collision attack applied to the implementation of the scalar multiplications can be applied to the ECDSA or ECDH in much the same way as described in Section 3.5. However, one cannot compare subtraces in the same way.

Using our example from Section 3.5, we can split an acquisition into subtraces

$$O = \{o_1, o_2, o_3, ...o_n\}$$

where o_i, for $1 \leq i \leq n$, correspond to the i-th subtrace taken while the i-th operation was being computed. If we use the double-and-add-always algorithm as an example then, O will be a series of addition and doubling operations

$$T = \{\delta_1, \alpha_1, \delta_2, \alpha_2 \ldots, \delta_n, \alpha_n\},$$

where a doubling operation δ_i and addition α_i for $i \in \{1, \ldots, n\}$.

We can compute mean traces for addition $\bar{\alpha}$ and doubling operations $\bar{\delta}$ using an entire set of acquisitions and subtract them from each subtrace, as previously, giving

$$\begin{aligned}\bar{T} &= \{\delta_1 - \bar{\delta}, \alpha_1 - \bar{\alpha}, \delta_2 - \bar{\delta}, \alpha_2 - \bar{\alpha}, \ldots, \delta_n - \bar{\delta}, \alpha_n - \bar{\alpha}\}, \\ &= \{\hat{\delta}_1, \hat{\alpha}_1, \hat{\delta}_2, \hat{\alpha}_2, \ldots, \hat{\delta}_n, \hat{\alpha}_n\}\end{aligned}$$

where we define $\hat{\delta}_i = \delta_i - \bar{\delta}$ and $\hat{\alpha}_i = \alpha - \bar{\alpha}$ for $1 \leq i \leq n$.

As, in our example in Section 3.5, we wish to compare all the operations in the third loop of the exponentiation algorithm with those in the fourth loop. Then we have comparisons

$$\{\hat{\delta}_3 - \hat{\delta}_4, \hat{\alpha}_3 - \hat{\alpha}_4, \hat{\delta}_3 - \hat{\alpha}_4, \hat{\alpha}_3 - \hat{\delta}_4\}.$$

The same operations can be compared using the method outlined in Section 3.5. However, when comparing an addition and a doubling operation we can note that these operations typically take different times to compute so one cannot subtract one subtrace from another. However, we can compare the field multiplications, each of which will take a fixed time to compute, for all operations.

In the trace T the doubling operations δ_i and additions α_i will have f and h field multiplications, respectively, where the values of f and h depend on the choice of algorithm and how points are represented. From each subtrace one can extract further subtraces representing the field multiplications.

$$\begin{aligned}\hat{\delta}_i &= \{d_{i,1}, \ldots, d_{i,f}\} \\ \hat{\alpha}_i &= \{a_{i,1}, \ldots, a_{i,h}\}\end{aligned}$$

for $i \in \{1, \ldots, n\}$.

One can then compare, for example, all of the field multiplications in a doubling operation with all of those in the following addition, giving an $f \times h$ element set

$$C_i = \{d_{i,1} - a_{i,1}, \ldots, d_{i,f} - a_{i,1}, d_{i,1} - a_{i,2}, \ldots, d_{i,f} - a_{i,2}, \ldots, d_{i,1} - a_{i,h}, \ldots, d_{i,f} - a_{i,h}\}$$

for $i \in \{1, \ldots, n\}$. Each C_i can be stored as a single trace and processed in the same way as a t-test where the subtraces are the same length. That is, we generate a set where the exponent bits are fixed and another where they are random, as described in Section 3.5 and compare the traces created by the above process. Again, to determine that there is no leakage, there should be no observe t-statistic greater than 4.5.

6 Conclusion

In this paper we describe how one can test implementations of public-key cryptographic algorithms using Test Vector Leakage Assessment (TVLA) initially proposed by Goodwill et al. [6]. Given the numerous choices that are made when implementing public-key cryptographic algorithms, the above can only act a guide to a side-channel evaluation and will need to be modified for each case.

References

[1] Kocher, P.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Koblitz, N., ed.: CRYPTO '96. Volume 1109 of LNCS., Springer (1996) 104–113

[2] Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In Wiener, M.J., ed.: CRYPTO '99. Volume 1666 of LNCS., Springer (1999) 388–397

[3] Quisquater, J.J., Samyde, D.: Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In Attali, I., Jensen, T.P., eds.: E-smart 2001. Volume 2140 of LNCS., Springer (2001) 200–210

[4] Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In Koç, C.K., Naccache, D., Paar, C., eds.: CHES 2001. Volume 2162 of LNCS., Springer (2001) 251–261

[5] Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In Joye, M., Quisquater, J.J., eds.: CHES 2004. Volume 3156 of LNCS., Springer (2004) 16–29

[6] Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side-channel resistance validation (September 2011)

[7] Schneider, T., Moradi, A.: Leakage assessment methodology—a clear roadmap for side-channel evaluations. In Güneysu, T., Handschuh:, H., eds.: CHES 2015. Volume 9293 of LNCS., Springer (2015) 495–513

[8] Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks — Revealing the Secrets of Smart Cards. Springer (2007)

[9] Mizuki, T., Hayashi, Y.: AES cipher keys suitable for efficient side-channel vulnerability evaluation. Cryptology ePrint Archive, Report 2014/770

[10] Montgomery, P.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48**(177) (1987) 243–264

[11] Joye, M., Yen, S.M.: The Montgomery powering ladder. In Jr., B.S.K., Ç. K. Koç, Paar, C., eds.: CHES 2002. Volume 2523 of LNCS., Springer (2003) 291–302

[12] Knuth, D.E.: The Art of Computer Programming. 2nd edn. Volume 2 / Seminumerical Algorithms. Addison-Wesley (1981)

[13] Hanley, N., Kim, H., Tunstall, M.: Exploiting collisions in addition chain-based exponentiation algorithms using a single trace. In Nyberg, K., ed.: CT-RSA 2015. Volume 9048 of LNCS., Springer (2015) 431–448

[14] Jaffe, J., Rohatgi, P., Witteman, M.: Efficient side-channel testing for public key algortihms: RSA case study (September 2011)