Attacks on Implementations of Cryptographic Algorithms: Side-Channel and Fault Attacks[*]

Erkay Savaş
Sabanci University
Istanbul, Turkey
erkays@sabanciuniv.edu

ABSTRACT

Cryptographic algorithms, which withstand cryptanalysis after years of rigorous theoretical study and detailed scrutiny have been shown to succumb to attacks that exploit the vulnerabilities in their implementations. Therefore, there has been a vast amount of research effort to find potential vulnerabilities in the implementation of cryptographic algorithms, and efficient and effective countermeasures if such vulnerabilities exist. In this paper, we survey side-channel and fault attacks, which are two powerful methods that have been demonstrated to render many implementations effectively broken. While we categorically analyze the attack techniques, possible countermeasures will also be discussed.

Categories and Subject Descriptors

E.3 [**Data Encryption**]: Code breaking, Public key cryptosystems, Standards (e.g., DES, PGP, RSA); C.2.0 [**Security and Protection**]: [Secure implementation of cryptographic algorithms]; H.2.7 [**Database Administration**]: [Security, integrity, and protection]

General Terms

Security

Keywords

Cryptography, cryptographic engineering, side-channel attacks, fault attacks

1. INTRODUCTION

The attacks that exploit the vulnerabilities in implementations of cryptographic algorithms can be analyzed in two categories: i) side-channel attacks and ii) fault attacks. Side-channel attacks exploit key-dependent variations in different executional aspects of cryptographic algorithms such as execution time [30, 31], power usage [31, 35], the number of cache hits/misses [41, 17, 38], the number of mispredicted branches [4, 2, 3], etc. These variations form an unintended *side-channel* that is observable by the adversary who can obtain information about the secret key used during the cryptographic operation. Side-channels are, in general, easily accessed in general-purpose computers where many processes share the same resources and in smart cards which depend on an outside power supply. For instance, other processes running on the same processor can collect information about the cryptographic process indirectly by timing its execution, the cache lines being accessed, etc. Similarly, a sophisticated smart card reader can monitor the power usage of a smart card since the card is powered by the reader.

Basic principle in side-channel countermeasures is to eliminate the dependency between the secret key used in cryptographic operation and its observable aspect. For instance, a straightforward method to thwart timing-based side-channel attacks is to implement cryptographic algorithms in such a way that all instances of cryptographic operations run in constant time. Similarly, simple power attacks can be prevented by eliminating key-dependent variations in the power usage of cryptographic operations. Avoiding the use of cache memory prevents any key-dependent cache misses/hits, hence cache-based side-channel attacks. We provide the detailed explanations about the attacks and possible countermeasures in subsequent sections.

In fault attacks [12, 10, 6, 22], the cryptographic device is forced to function incorrectly resulting in a faulty result, which can be exploited to reveal the secret key used in cryptographic computation. Flipping a bit or bits in a register or combinational logic during the cryptographic computation (*fault injection*) will result in faulty results. An adversary, who has direct access to cryptographic device, can inject faults by manipulating the working conditions of cryptographic device such as introducing a glitch on supply voltage, exposing it to temperatures outside of normal working conditions, changing clock rate, or exposing it to a radiation source (e.g., laser, X-ray, etc.) [49, 24]. As shown in literature [50], powerful adversaries can induce faults in cryptographic devices with high precision in time and space, which indicates the difficulty of protecting secret keys against fault attacks.

A particular implementation method based on the Chinese Remainder Theorem (CRT) for accelerating RSA decryption operation [43] was shown to be vulnerable to fault attacks which can be relatively easily applied. Similarly, Biham and Shamir [10] demonstrated that a bit flip introduced in the last round of DES algorithm helps reduce the key space.

A simple remedy for fault attacks is to check the result for correctness before outputting it. The same cryptographic computation is executed more than once possibly in separate, identical units that are supposed to be producing the same output. A discrepancy in the results indicate the presence of a fault, possibly an attack. If adversary achieves introducing the same error in multiple identical computations, then this countermeasure is useless. However, introducing the same error in multiple locations in a cryptographic device is difficult. On the other hand, doing the same computation more than once can be expensive. Therefore, less expensive error-detection codes are proposed to be used as a countermeasure where parity bits are used to detect a fault. However, errors introduced by an intelligent adversary may not be detected by classical error detection codes, which are usually effective against random errors introduced by mother nature. A different class of error-detection codes and their implementations are proposed against faults that can be induced by an intelligent adversary with high precision [28, 32, 20, 33, 56, 58, 59].

In subsequent sections, we give examples of effective fault attacks and proposed countermeasures which can be profitably used as an effective and efficient protection mechanism.

2. SIDE-CHANNEL ATTACKS

In this section, we explain different side-channel attacks and countermeasures.

2.1 Power Attacks

In this section, power attacks against RSA algorithm are discussed. Power attacks use so-called *power trace* which keeps track of power usage variations during a cryptographic computation. RSA algorithm uses modular exponentiation as the primitive operation. Binary left-to-right algorithm, given in Algorithm 1, is one of the most efficient methods to calculate modular exponentiation with big integers. In

```
Algorithm 1 Binary left-to-right modular exponentiation
Require: $m$: message, $d = d_{k-1}, \ldots d_1, d_0$: private exponent, $N$: Modulus, $k = \lceil \log_2(N) \rceil$
Ensure: $s = m^d \bmod N$
  $s \leftarrow 1$
  for $i = k - 1$ to $0$ do
    $s \leftarrow s \times s \bmod N$
    if $d_i = 1$ then
      $s \leftarrow s \times m \bmod N$
    end if
  end for
```

Algorithm 1, the modular multiplication, $s \times m \bmod N$ is key dependent. When the corresponding key bit d_i is nonzero this operation is performed, while the modular squaring step $s \times s \bmod N$ is always executed independent of d_i. Modular multiplication is more complicated than modular squaring, therefore generally more power is required to perform it. An attacker who can monitor and record the power usage during the computation with the right equipment can infer when a modular multiplication is performed. This indicates that the corresponding bit of the private exponent d is 1. This attack method is referred as *simple power analysis* (SPA). Effective countermeasures can also be easily applied.

The problem with the SPA is that the effect of secret key on power trace of cryptographic computation can be difficult to observe due to noise. A more powerful method known as the *differential power analysis* (DPA) [31, 36, 26] can be effectively applied to both symmetric key and public key algorithms. The DPA relies on statistical techniques to eliminate the effect of noise on the observations in the side-channel.

The basic idea of the method is to guess relatively small section of secret key bits used during computation (e.g., several key bits entering an s-box, an individual bit of the private exponent in RSA). A selection function D is defined to partition power measurements into two alternate sets, S_1 and S_0 depending on the guessed values of selected bits. For instance, the selection function is used predict power consumption for a particular operation, in which guessed bits are used (e.g., an s-box operation, modular multiplication $s \times m \bmod N$). If the selection function predicts high power consumption, the measured power signal is placed into a set, S_1, otherwise it is placed into the alternate set, S_0. In other words, power measurements are partitioned into two sets based on their predicted power consumptions for the guessed values of key bits. If the guessed bits are correct, then two sets, S_0 and S_1 can be statistically distinguishable. For instance, the averages of the two sets will be discernibly different from each other if our guess is correct provided that sufficient measurement results are obtained. Otherwise, the two sets will show no statistical difference from each other.

If the guess turns out to be incorrect, a similar partitioning is carried out for another guess for the values of the targeted key bits. From this discussion, one should understand that the number of the targeted key bits is selected in such a way that trying out their all possible values is computationally feasible.

2.2 Timing Attacks

Execution times of cryptographic computations show variations, which can be exploited to infer secret keys used during computation. As an example, time to execute Algorithm 1 may leak the number of nonzero bits in the private exponent d. More sophisticated algorithms, which can reveal the bits of d, require details of the implementation. In this section, one particular implementation of modular exponentiation in RSA will be shown to be vulnerable to timing attacks.

Modular exponentiation is nothing but repeated modular multiplications (and squaring operations) and the Montgomery multiplication method [37, 15] is one of the most efficient algorithms used in cryptographic applications. A description of the Montgomery algorithm is given in Algorithm 2. As can be seen in the algorithm, the Montgomery

Algorithm 2 Montgomery modular multiplication

```
Require: $s\ m$, $N$, $k = \lceil \log_2(N) \rceil$
Ensure: $s = s \times m \times 2^{-k} \bmod N$
  $s \leftarrow s \times m$
  $s \leftarrow (s - (s \times N^{-1} \bmod 2^k) \times N)/2^k$
  if $s > N$ then
    $s \leftarrow s - N$
  end if
```

algorithm does not compute modular multiplication directly and extra operations are needed to obtain the desired result. However, the overhead due to extra operation becomes negligible when the algorithm is used in modular exponentiation. The advantage of the Montgomery algorithm is due to fact that it substitutes expensive division operation with division by two's powers, which is nothing but, shift operation in digital systems. The timing attack utilizes the overhead of the final subtraction operation in Algorithm 2 (i.e., $s \leftarrow s - N$) when the multiplication operation (i.e., $s \leftarrow s \times s \bmod N$ in Algorithm 1) is executed during modular exponentiation.

The attack assumes the knowledge of the most significant $i-1$ bits of the private exponent, $d_{k-1}, \dots d_{k-i-1}$, and guesses the next bit d_{k-i}. Since these bits of the private exponent are known, one can compute the value of s before the i-th iteration of the exponent. Consequently, depending on the guess for d_{k-i} one can predict whether the final subtraction operation is needed. For instance, if the guess is $d_{k-i} = 1$, one can predict whether the final subtraction is needed. When the final subtraction is needed, we expect a slight increase in the execution time of the exponentiation, which is usually hard to discern in one execution of the algorithm.

For different input messages, the exponentiation algorithm is executed and the running time is recorded along with the corresponding input message. Then, for every message, we predict whether the final subtraction is needed in the i-th iteration. If the final subtraction is needed the running time is placed into a set, S_1, otherwise it is placed into the alternate set S_0. If the guess for d_{k-i} is correct we expect that average[1] value of timing measurements in S_1 is distinguishably larger than that of S_0 provided that sufficient number of timing measurements are collected for different input messages. If the guess is incorrect, we will see no difference in the sets S_0 and S_1.

Depending on the noise level in the cryptographic device during the computation or the relative importance of time variation due to final subtraction in the overall computation, high number of time measurements is needed.

A similar attack is also possible when the power consumption is recorded instead of execution time. Extra operations due to a final subtraction operation will increase the power consumption during the exponentiation operations, which in turn leads to a power attack. Works such as [55, 23] provide new algorithms for Montgomery multiplication without final subtraction to eliminate the possibility of the described attacks.

[1]Other statistical values such as variance can also be used.

2.3 Cache Attacks

Cache attacks [30, 29, 40, 52, 41, 17, 39, 13, 11, 44, 51] use the fact that cache memory is shared by many processes running concurrently in the processor core and that it is possible to monitor cache activities of a cryptographic process indirectly. If the cache activity of a cryptographic process depends on the secret key, then certain information about the secret key can be inferred by monitoring the cache access patterns. Using different methods and tools, it is sometimes possible to learn whether a particular cache access is a hit or miss, the number of hits/misses, or whether a particular cache line is accessed during cryptographic operations. Depending on the observed cache activity we can mount various cache-based side-channel attacks.

Since many cache attacks are developed for the AES algorithm [18], which is the current data encryption standard by the NIST, we base our discussions on the AES implementations, which are known to be vulnerable to cache attacks. A particular implementation of the AES makes an extensive use of lookup tables to accelerate s-box operations, which fit in the first level cache of modern day processors. Since caches are shared by many concurrently executing processes, lookup operations during cryptographic calculations can result in cache misses due to the cache contention created by other processes (i.e., the AES lookup table entries can be evicted by another process which wants to use the same cache lines). These contentions can be caused accidentally by innocuous processes or deliberately by ill-intentioned spy processes deployed by the adversary.

Cache attacks can be grouped into three categories depending on the kind of exploited cache activity. These categories are explained in the following sections.

2.3.1 Access-Driven Cache Attacks

Access-driven attacks exploit the information as to whether a cache set is accessed or not during the cryptographic operation. To mount the attack, a spy process is deployed by the attacker to determine which cache sets are accessed or not by the cryptographic process. In [51], a spy process scheduled between two consecutive executions of the cryptographic process is employed first to evict all cache sets and then identify the cache sets that are accessed during the cryptographic operation. The proposed attack assumes that plaintext or ciphertext are known.

For instance, the access-driven attack to the last round of the AES as described in [47] uses a key elimination strategy and unaccessed cache sets during the AES encryption. The indexes used in the last round to access the lookup tables can be written in terms of known ciphertext and unknown round key bytes, i.e., $\mathcal{S}^{-1}(c_i \oplus k_i^{10})$ for $i = 0, \dots, 15$, where $\mathcal{S}^{-1}$ is the inverse s-box operation, and c_i and k_i^{10} are ciphertext and key bytes of the tenth round, respectively. If we know a certain cache set is not accessed we can obtain an inequality such as $< \mathcal{S}^{-1}(c_i \oplus k_i^{10}) > \neq y$, which helps eliminate wrong candidates for k_i^{10}. Here, y stands for the index of unaccessed cache set and the $<>$ symbol represents certain number of most significant bits of the argument inside, which is determined by the cache line size, the table size, and size of the table entries. For more information, the reader is referred to [47].

In the key elimination strategy, each inequality eliminates many incorrect key candidates. For instance if $<>$ symbol represents the upper nibble of the byte, and y is a four bit number, one inequality eliminates 16 wrong candidates out of 255. Apparently, we can increase the possibility of eliminating all 255 wrong key candidates if we make many observations of unaccessed cache sets.

2.3.2 *Trace-Driven Cache Attacks*

Trace-driven cache attack [40] assumes that a trace of cache hits and misses during a series of lookup table accesses is available to the attacker. For example, the attacker knows the outcomes of the first two accesses in the last round of AES in terms of cache hits and misses. If we assume that the cache is flushed by a spy process before the encryption, there will be two possibilities for the first two accesses, namely $\{MM, MH\}$, where M and H stand for miss and hit, respectively. A trace can be obtained by observing power or electro-magnetic emissions of cryptographic device which is assumed to be in full control of the attacker. Trace-driven attacks are investigated in detail in [1, 19, 61].

To illustrate the attack, assume that we have the trace $\{MM\}$ for the first two accesses in the last round of the AES. This means that the first two accesses are to two different cache sets. Then, we can conclude that $\mathcal{S}^{-1}(c_i \oplus k_i^{10}) \neq \mathcal{S}^{-1}(c_j \oplus k_j^{10})$, where i and j represent the indices of ciphertext and key bytes used in the first and second lookup table accesses in the last round, respectively. Assuming the ciphertext bytes are known, one such inequality is expected to eliminate 2^{12} wrong candidates for k_i^{10} and k_j^{10} out of a total of $2^{16} - 1$ wrong candidates. If the trace is $\{MH\}$, then we obtain the equality $\mathcal{S}^{-1}(c_i \oplus k_i^{10}) = \mathcal{S}^{-1}(c_j \oplus k_j^{10})$. And this equality is expected to eliminate $2^{16} - 2^{12}$ wrong candidates for the same key bytes.

To eliminate all values for other key bytes we need to collect many and longer traces. For example, to learn all key bytes used in the tenth round of the AES, we need traces of length 16.

2.3.3 *Time-Driven Cache Attacks*

In cache-based time-driven attacks [13, 53], attacker exploits the variations in execution times of cryptographic operations due to the number of cache hits or misses. For instance, when an AES lookup table entry is not found in the cache memory, the access is directed to the main memory (or lower level of cache memory if one exists), which takes tens to hundreds of clock cycles. If number of misses (or hits) during cryptographic operation is accessible by some means (e.g., through the hardware performance counters available in many general-purpose microprocessors), the attack can be applied using this information more accurately [53].

A simple time-driven attack on the AES can be explained as follows. For simplicity, we utilize the number of hits, which is correlated with execution time, and assume that the cache is flushed before every encryption. While many attack possibilities exists, we can focus on any two lookup table accesses in the last round of AES. Two bytes of round keys are used in the calculation of indexes for these two accesses; namely we use the following expressions as indexes, $\mathcal{S}^{-1}(c_i \oplus k_i^{10})$ and $\mathcal{S}^{-1}(c_j \oplus k_j^{10})$. Assuming that ciphertext bytes are known, we start guessing the key bytes k_i^{10} and k_j^{10}. Based on the guessed values of the key bytes, we can predict whether the second access is a hit or miss (e.g., if $< \mathcal{S}^{-1}(c_i \oplus k_i^{10}) > = < \mathcal{S}^{-1}(c_j \oplus k_j^{10}) >$, it is a hit) during the encryption. Similar to the timing attacks explained previously, we form two sets of number of hits, S_1 and S_0. If the prediction is a hit, the number of hits measured during actual encryption operation is placed in the set S_1, otherwise in S_0. Furthering the analogy to the timing attacks, our prediction of a cache access is used as a selection function.

If our guess is correct, then the measurements in S_1 are expected to be higher than those in S_0. If the guess is not correct, the two sets will show no statistical difference. The total number of guesses for k_i^{10} and k_j^{10} is 2^{16}; a feasible number for an exhaustive search. A simple scoring technique based on the difference between the average number of hits in these two sets will order key guesses for their likelihood of being the correct key. The key with the highest score is expected to be the correct key if the sets S_1 and S_0 are populated with sufficient number of measurements. For more information, reader is referred to [46].

Another timing attack that can be categorized as timing attack is known as Bernstein attack [17], which is intended to be applied remotely on a server performing AES encryptions for clients connected to the server (e.g., OpenSSL server). It is originally applied in a scenario, in which the AES server and client are running in the same processor, and therefore it is not known whether it can be applied remotely. Some experiments are conducted in [5] to successfully apply it remotely, such as the scenario where the server and the client are running on different cores in a multi-core computer. To the best of our knowledge, the attack has never been successfully applied over a network. For the detailed analysis for the feasibility of timing attacks in general, reader is referred to [16].

Another important aspect of Bernstein attack worth investigating is its cause. Neve [38] explains the cause as the other system processes creating contentions with cryptographic process resulting in cache misses during AES lookup operations. If attacker *learns* which cache locations are the subject of contentions, he can mount a successful attack and infer secret key bits. In [5], it is found out that the other processes causing contentions in cache memory are in fact kernel processes responsible for client-server communication on behalf of AES server process. This finding is especially important since the contentions are inadvertently created among different parts of the server implementation. Therefore, one may also need to check unintended cache contentions during the software development process.

2.4 Counter Measures

The most efficient countermeasures against timing attacks are constant time implementations of cryptographic algorithms, whereby the execution time does not vary with inputs. Avoiding the use of lookup tables eliminates all cache-based attacks. In the case of the AES, the first and the last two rounds are especially vulnerable to cache attacks, therefore their cache-free implementations provide substantial protection. Intel's AES-NI set [25] contains specialpurpose instructions, which not only accelerate AES encryption, but also protect it against cache attacks. However, due to the fact that not all microprocessors have similar architectural support and that using lookup tables is an integral part of acceleration techniques in many other cryptographic algorithms, cache attacks are still relevant.

The Montgomery Powering Ladder [27] is an effective method for private key-independent modular exponentiation algorithm. Randomization [21] that prevents predictions in timing attacks is another effective method. Masking [42, 45], which randomizes the sensitive intermediate values during cryptographic computation, thus eliminating the correlation between power consumption and these intermediate values, is an effective countermeasure for power attacks.

3. FAULT ATTACKS

Targeting a register or transistor, the adversary aims to introduce a bit-flip that will cause faulty computation. Depending on the capabilities of the attacker, bit flips can be introduced at different granularity, individual bits, a group of bits, etc. In this section, two well-known fault attacks are explained and relevant countermeasures are discussed. Interested readers are referred to [54] for more information on fault attacks and countermeasures.

3.1 Fault Attack Against RSA-CRT Implementation

RSA decryption operation is simply a modular exponentiation of big integers, i.e., $s = m^d \pmod N$, where d is the private key, N is the modulus, factored into two private random primes, p and q, i.e. $N = pq$. The owner, who knows the prime factors of N, can accelerate the modular exponentiation using the Chinese Remainder Theorem (CRT). Instead of expensive $m^d \bmod N$, the user computes the following exponentiations with half size operands:

$$s_p = m_p^{d_p} \pmod p \text{ and } s_q = m_q^{d_q} \pmod q,$$

where $m_p = m \pmod p$, $m_q = m \pmod q$, $d_p = d \pmod{p-1}$, and $d_q = d \pmod{q-1}$. Then, the user combines them to obtain the final result using the CRT

$$s = (s_p \cdot q \cdot (q^{-1} \bmod p) + s_q \cdot p \cdot (p^{-1} \bmod q)) \pmod N.$$

Due to the quadratic complexity of involved arithmetic operations, the new computation technique proves to be much more efficient. However, if attacker disturbs the computation of s_p (or s_q) the CRT step produces the output $s' \neq s$. Let us assume that s'_p is the result of faulty computation while the s_q is correct result and $s' = CRT(s'_p, s_q)$. Then, we have

$$s' \equiv s \bmod q \Rightarrow s' - s \equiv 0 \bmod q \Rightarrow s' - s = kq \qquad (1)$$

$$s' \not\equiv s \bmod p \Rightarrow s' - s \not\equiv 0 \bmod p \Rightarrow s' - s \neq tp \qquad (2)$$

If the adversary obtains both the correct and faulty results, namely s and s', then it can obtain one of the factors of N by computing $\gcd(s' - s, N) = q$. This is true since both $s' - s$ and N are a multiple of q and the former is not a multiple of p. Also, Lenstra [34] demonstrated that one does not need the correct result and using the following relation suffices to give one of the factors, $\gcd((s')^e - m, N) = q$, where e is the public exponent. Consequently, one incorrect execution of RSA decryption operation can break the RSA.

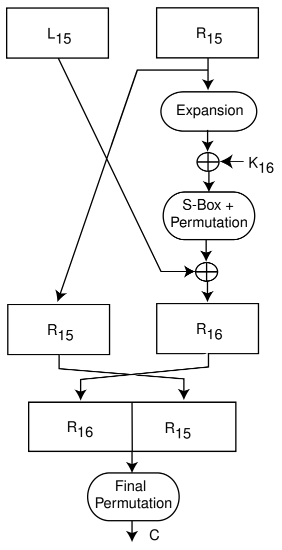

Figure 1: Last round of DES

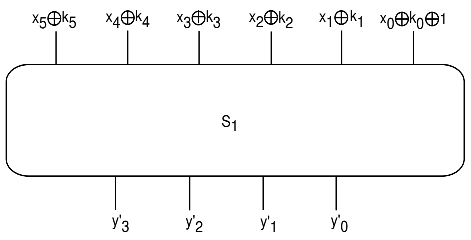

Figure 2: Attacking s-box S1 in the last round of DES

3.2 Differential Fault Attack Against DES

A powerful attack introduced by Biham and Shamir [10] utilizes differential properties of the s-boxes of the DES algorithm. The attack can be illustrated by its instance based on flipping one of the input bits of one of the s-boxes.

The last round operations of the DES are illustrated in Figure 1, where L_{15} and R_{15} are left and right hand sides of the 64-bit DES block, respectively, before the last round. If an error e is introduced in R_{15} (i.e., $R'_{15} = R_{15} \oplus e$), the faulty ciphertext $FC \neq C$ is observed at the output of the DES. Assuming we have both the correct and faulty ciphertexts, we can compute the error as $e = R'_{15} \oplus R_{15}$. Also, the error at the output of the s-box layer can be computed in a similar way, i.e., from $\tilde{e} = R'_{16} \oplus R_{16}$.

For example, assume that one bit error is induced at the input of the s-box S_1 in the last round of the DES as illustrated in Figure 2, where k_i stands for the bits of the round key. When $\tilde{\epsilon} = y \oplus y'$ and $\epsilon = x \oplus x'$ are known, from the XOR distribution table of S_1 one can find the set of candidates for the input of S_1. Assuming that the number of candidates are few, one can eliminate many candidates for the corresponding bits of the round key since x can be calculated from the ciphertext. For instance, if $\tilde{\epsilon} = 5$ and $\epsilon = 1$, then there are only two possible values of input of s-box S_1 from its XOR distribution table. These values are 6 and 7.

Thus, this attack eliminates many candidates out of 64 for the key bits $(k_5, k_4, k_3, k_2, k_1, k_0)$.

Apparently, flipping one bit in the last round is not sufficient to find all keys, the bit flipping should be introduced in multiple locations and the attack must be repeated before the full discovery of the secret key.

3.3 Feasibility of Fault Attacks and Countermeasures

Difficulty of applying a particular fault attack depends on whether a bit flipping can be introduced in the right location at the right time, which is referred as time and space precision in the attack. Depending on the capabilities of the adversary the desired results can be achieved. A powerful adversary, to whom expensive equipments are available, can introduce errors in multiple locations with high accuracy. Also an attacker can repeat the attack millions of times to achieve desired results. Recent works [50, 9, 8] suggest that powerful attackers need to be taken into account when the countermeasures are designed.

On the other hand, while some attacks require high accuracy in the location and time of the fault, other attacks can work with much lower levels of accuracy. For instance, in the attack against the CRT implementation of RSA, any fault during the computation of $s_p = m_p^{d_p} \bmod p$ produces the desired result. Since this computation takes relatively long time and the exponentiation unit is a considerably large circuit, where any bit flipping is useful, the attack is highly feasible.

On the other hand, differential fault attack is harder to apply since more precision is needed in terms of both location and time. For instance, a bit flipping which is intended to be introduced in a particular round requires much more accuracy since a DES round is executed much faster than modular exponentiation and requires much less chip space to implement. Whether a countermeasure should be implemented, or the effectiveness of adopted countermeasure depends the feasibility of the attack. Since efficiency of implementations is almost always a prominent requirement, the resources used for countermeasures should commensurate with the feasibility and effectiveness of the attack.

Countermeasures based on redundancy are always plausible solutions to fault attacks. Performing the same cryptographic computation more than once and checking the consistency among multiple outputs is an easy method to implement. Alternatively, after an encryption, decryption of the ciphertext can be used to check whether the original message is obtained before giving out any output. For instance, performing same DES encryption twice is a viable solution since DES encryption can be executed very fast and hardware implementation of DES algorithm is low cost one can resort to double encryption. The attacker's challenge is now introducing the exact same fault in both executions, which is usually difficult.

On the other hand, naive redundancy approach, as explained for symmetric ciphers may not be a feasible solution for public key algorithms such as RSA. Instead, low cost solutions are needed since public key operations are expensive both in terms of time and space complexity. Shamir [48] proposed a simple solution based on a method that uses a small integer t and computes

$$s_{pt} = m_p^{d_p} \bmod p \cdot t \text{ and } s_{qt} = m_q^{d_q} \bmod q \cdot t.$$

Then, the check $s_{pt} \equiv s_{qt} \bmod t$ before applying the CRT to calculate s. As it is shown in [7], this method does not protect the entire computation and thus a fault attack targeted at CRT computation can still break the RSA. More sophisticated countermeasures are discussed in [21]. However, these countermeasures are specific to the CRT implementation of the RSA algorithm and not all can be directly used for other public key algorithms.

A more generic approach for protecting cryptographic algorithms against fault attacks relies on protecting the arithmetic operations, on which most of cryptographic computations' time are spent. Protecting addition and multiplication operations using non-linear arithmetic error-correcting codes is suggested in [20, 56, 58, 59]. This way, a wider range of public key algorithms based on modular arithmetic can be protected against fault attacks. The cost of integrating non-linear arithmetic error-correcting codes in embedded processors is evaluated in [57, 60]. Detecting capability and design choices of the error correcting codes against an intelligent and powerful attacker are discussed in detail in [59].

Various categorizations for attacks as well as countermeasures are given in [54]. Protections can be achieved at protocol, cryptographic primitive, and arithmetic levels depending on the type of attack. Another categorization is given depending on the part of the processor, which is targeted by the attack: integrity check of input parameters, parallel redundant computations (e.g., parity and error-detection codes), checking inherent algorithm properties (e.g., verification check for signature generation algorithms).

4. CONCLUSIONS

Implementing cryptographic algorithms, either in hardware or software, is not an endeavor for unexperienced developers without in-depth knowledge of side-channel and fault attacks, their combination, and effective countermeasures. Therefore, a major field in *cryptographic engineering*, an important discipline emerged within cryptology, aims to address security problems associated with implementations of cryptographic algorithms. In this paper, we provide a broad overview of two attack categories, their causes, specific attack techniques keeping the discussions as simple as possible so that they are accessible to layperson in the field.

We provide an extensive bibliography to refer interested readers to relevant sources for a specific attack technique, and effective and efficient countermeasures. Side-channel and fault attacks are perhaps the most vibrant field in the area of applied cryptography and there is a plethora of scientific publications, conferences, workshops, and other events, where recent results are presented. Therefore, any survey that intends to provide an exhaustive bibliography for the attacks and countermeasures is not likely to succeed. This paper is intended to raise the awareness on the issue and also to be an entry point for further reading to the novice in the field.

5. ACKNOWLEDGMENTS

This work is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under project number 110E037.

6. REFERENCES

[1] O. Aciiçmez and Ç. K. Koç. Trace-driven cache attacks on AES (short paper). In P. Ning, S. Qing, and N. Li, editors, *ICICS*, volume 4307 of *Lecture Notes in Computer Science*, pages 112–121. Springer, 2006.

[2] O. Aciiçmez, Ç. K. Koç, and J.-P. Seifert. On the power of simple branch prediction analysis. *IACR Cryptology ePrint Archive*, 2006:351, 2006.

[3] O. Aciiçmez, S. Gueron, and J.-P. Seifert. New branch prediction vulnerabilities in openssl and necessary software countermeasures. *IACR Cryptology ePrint Archive*, 2007:39, 2007.

[4] O. Aciiçmez, J.-P. Seifert, and Ç. K. Koç. Predicting secret keys via branch prediction. *IACR Cryptology ePrint Archive*, 2006:288, 2006.

[5] A. C. Atici, C. Yilmaz, and E. Savaş. An approach for isolating the sources of information leakage exploited in cache-based side-channel attacks. Workshop on Trustworthy Computing, Washington, D.C., USA, 18-20 June, 2013. to appear.

[6] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault attacks on rsa with crt: Concrete results and practical countermeasures. In B. S. Kaliski, Ç. K. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 260–275, 2002.

[7] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault attacks on rsa with crt: Concrete results and practical countermeasures. In B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2002.

[8] J. Balasch, B. Gierlichs, and I. Verbauwhede. An in-depth and black-box characterization of the effects of clock glitches on 8-bit MCUs. In Breveglieri et al. [14], pages 105–114.

[9] R. Bekkers and H. König. Fault injection, a fast moving target in evaluations. In Breveglieri et al. [14], page 65.

[10] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO 97*, volume 1294 of *LNCS*, pages 513–525, 1997.

[11] J. Blömer and V. Krummel. Analysis of countermeasures against access driven cache attacks on AES. In C. M. Adams, A. Miri, and M. J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *LNCS*, pages 96–109. Springer, 2007.

[12] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In W. Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[13] J. Bonneau and I. Mironov. Cache-collision timing attacks against AES. cryptographic hardware and embedded systems. In *Lecture Notes in Computer Science series 4249*, pages 201–215. Springer, 2006.

[14] L. Breveglieri, S. Guilley, I. Koren, D. Naccache, and J. Takahashi, editors. *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011*. IEEE, 2011.

[15] Ç. K. Koç, T. Acar, and B. S. K. Jr. Analyzing and comparing montgomery multiplication algorithms. *IEEE MICRO*, 16(3):26–33, 1996.

[16] S. A. Crosby, D. S. Wallach, and R. H. Riedi. Opportunities and limits of remote timing attacks. *ACM Trans. Inf. Syst. Secur.*, 12(3):17:1–17:29, Jan. 2009.

[17] D. Bernstein. Cache-Timing Attacks on AES. Website, 2005. `http://cr.yp.to/papers.html#cachetiming`.

[18] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[19] J.-F. Gallais, I. Kizhvatov, and M. Tunstall. Improved trace-driven cache-collision attacks against embedded AES implementations. In *Proceedings of the 11th international conference on Information security applications*, WISA'10, pages 243–257, Berlin, Heidelberg, 2011. Springer-Verlag.

[20] G. Gaubatz, B. Sunar, and M. G. Karpovsky. Non-linear residue codes for robust public-key arithmetic. In *FDTC*, pages 173–184, 2006.

[21] C. Giraud. An rsa implementation resistant to fault attacks and to simple power analysis. *IEEE Trans. Computers*, 55(9):1116–1120, 2006.

[22] C. Giraud and H. Thiebeauld. A survey on fault attacks. In J.-J. Quisquater, P. Paradinas, Y. Deswarte, and A. A. E. Kalam, editors, *CARDIS*, pages 159–176. Kluwer, 2004.

[23] G. Hachez and J.-J. Quisquater. Montgomery exponentiation with no final subtractions: Improved results. In Çetin Kaya Koç and C. Paar, editors, *CHES*, volume 1965 of *Lecture Notes in Computer Science*, pages 293–301. Springer, 2000.

[24] H. B.-E. Hamid, H. Choukri, D. N. M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[25] J. R. (Intel). Intelő advanced encryption standard instructions (aes-ni). Website, last accessed on March 31, 2013. `http://http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni/`.

[26] J. Jaffe. A first-order dpa attack against AES in counter mode with unknown initial counter. In *Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, CHES '07, pages 1–13, Berlin, Heidelberg, 2007. Springer-Verlag.

[27] M. Joye and S.-M. Yen. The montgomery powering ladder. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '02, pages 291–302, London, UK, UK, 2003. Springer-Verlag.

[28] M. G. Karpovsky, K. J. Kulikowski, and A. Taubin. Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard. In *DSN'04*, pages 93–101, 2004.

[29] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side channel cryptanalysis of product ciphers. *J. Comput. Secur.*, 8(2,3):141–158, Aug. 2000.

[30] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *CRYPTO*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[31] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[32] A. T. Konrad J. Kulikowski, Mark G. Karpovsky. Robust codes for fault attack resistant cryptographic hardware. In *FDTC*, pages 1–12, 2005.

[33] K. J. Kulikowski, Z. Wang, and M. G. Karpovsky. Comparative analysis of fault attack resistant architectures for private and public key cryptosystems. In *FDTC*, pages 41–50, 2008.

[34] A. K. Lenstra. Memo on RSA signature generation in the presence of faults. Technical report, EPFL, 1996. manuscript.

[35] S. Mangard. A simple power-analysis (spa) attackon implementations of the AES key expansion. In P. J. Lee and C. H. Lim, editors, *ICISC*, volume 2587 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2002.

[36] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In Çetin Kaya Koç and C. Paar, editors, *CHES*, volume 1717 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 1999.

[37] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):pp. 519–521, 1985.

[38] M. Neve. *Cache-based Vulnerabilities and SPAM analysis*. PhD thesis, Universite catholique de Louvain, 2006.

[39] M. Neve, J.-P. Seifert, and Z. Wang. A refined look at bernstein's AES side-channel analysis. In F.-C. Lin, D.-T. Lee, B.-S. P. Lin, S. Shieh, and S. Jajodia, editors, *ASIACCS*, page 369. ACM, 2006.

[40] D. Page. Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel. Technical Report CSTR-02-03, Department of Computer Science,University of Bristol, June 2002.

[41] C. Percival. Cache missing for fun and profit. In *Proc. of BSDCan 2005*, 2005.

[42] T. Popp, S. Mangard, and E. Oswald. Power analysis attacks and countermeasures. *IEEE Design & Test of Computers*, 24(6):535–543, 2007.

[43] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for rsa public-key cryptosystem. *Electronics Letters*, 18(21):905–907, 1982.

[44] C. Rebeiro, M. Mondal, and D. Mukhopadhyay. Pinpointing cache timing attacks on AES. In *VLSI Design*, pages 306–311. IEEE, 2010.

[45] M. Rivain, E. Prouff, and J. Doget. Higher-order masking and shuffling for software implementations of block ciphers. In *LNCS*, CHES '09, pages 171–188, Berlin, Heidelberg, 2009. Springer-Verlag.

[46] E. Savaş and C. Yilmaz. A comprehensive analysis of cache attacks: A case study for aes. submitted for publication. 2013.

[47] E. Savas and C. Yilmaz. Cache attacks: An information and complexity theoretic approach. In A. Levi, M. Badra, M. Cesana, M. Ghassemian, Ö. Gürbüz, N. Jabeur, M. Klonowski, A. Maña, S. Sargento, and S. Zeadally, editors, *NTMS*, pages 1–7. IEEE, 2012.

[48] A. Shamir. Improved method and apparatus for protecting public key schemes from timing and fault attacks. International Patent Number: WO 98/52319, Nov. 1998.

[49] S. P. Skorobogatov and R. J. Anderson. Optical fault induction attacks. In B. S. Kaliski, Ç. K. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 2–12, 2003.

[50] E. Trichina and R. Korkikyan. Multi fault laser attacks on protected crt-rsa. In L. Breveglieri, M. Joye, I. Koren, D. Naccache, and I. Verbauwhede, editors, *FDTC*, pages 75–86. IEEE Computer Society, 2010.

[51] E. Tromer, D. Osvik, and A. Shamir. Efficient cache attacks on AES, and countermeasures. *Journal of Cryptology*, 23:37–71, 2010.

[52] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeria, and H. Miyauchi1. Cryptanalysis of DES Implemented on Computers with Cache. In C.D. Walter et al., editor, *CHES 2003 LNCS*, volume 2279, pages 62–76, 2003.

[53] L. Uhsadel, A. Georges, and I. Verbauwhede. Exploiting hardware performance counters. In *Fault Diagnosis and Tolerance in Cryptography, 2008. FDTC '08. 5th Workshop on*, pages 59 –67, aug. 2008.

[54] I. Verbauwhede, D. Karaklajic, and J. Schmidt. The fault attack jungle - a classification model to guide you. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2011 Workshop on*, pages 3–8, 2011.

[55] C. Walter. Montgomery exponentiation needs no final subtractions. *Electronics Letters*, 35(21):1831–1832, 1999.

[56] Z. Wang, M. Karpovsky, B. Sunar, and A. Joshi. Design of reliable and secure multipliers by multilinear arithmetic codes. In S. Qing, C. Mitchell, and G. Wang, editors, *ICICS 2009*, volume 5927 of *LNCS*, pages 47–62, 2009.

[57] K. Yumbul, S. S. Erdem, and E. Savaş. Design and implementation of robust embedded processor for cryptographic applications. In O. B. Makarevich, A. Elçi, M. A. Orgun, S. A. Huss, L. K. Babenko, A. G. Chefranov, and V. Varadharajan, editors, *SIN*, pages 178–185. ACM, 2010.

[58] K. Yumbul, S. S. Erdem, and E. Savas. On protecting cryptographic applications against fault attacks using residue codes. In Breveglieri et al. [14], pages 69–79.

[59] K. Yumbul, S. S. Erdem, and E. Savas. On selection of modulus of quadratic codes for the protection of cryptographic operations against fault attacks. *IEEE Transactions on Computers*, 99(PrePrints):1, 2012.

[60] K. Yumbul, E. Savas, O. Kocabas, and J. Grossschaedl. Design and implementation of a versatile cryptographic unit for risc processors. *Security and Communication Networks*, pages n/a–n/a, 2012.

[61] X.-J. Zhao and T. Wang. Improved cache trace attack on AES and CLEFIA by considering cache miss and s-box misalignment. *IACR Cryptology ePrint Archive*, pages 56–56, 2010.