RSA Key Recovery from Digit Equivalence Information

Chichanok Chuengsatiansup[1], Andrew Feutrill[1,2], Rui Qi Sim[1], and Yuval Yarom[1]

[1] The University of Adelaide, Australia
[2] CSIRO, Data61, Australia

Abstract. The seminal work of Heninger and Shacham (Crypto 2009) demonstrated a method for reconstructing secret RSA keys from partial information of the key components. In this paper we further investigate this approach but apply it to a different context that appears in some side-channel attacks. We assume a fixed-window exponentiation algorithm that leaks the *equivalence* between digits, without leaking the value of the digits themselves.
We explain how to exploit the side-channel information with the Heninger-Shacham algorithm. To analyse the complexity of the approach, we model the attack as a Markov process and experimentally validate the accuracy of the model. Our model shows that the attack is feasible in the commonly used case where the window size is 5.

1 Introduction

One of the roles of a cryptographer is to ensure that implementations of cryptographic primitives are secure. In recent decades, side-channel attacks have been identified as a major threat to the security of cryptographic implementations. These attacks observe the effects that executing implementation of a cryptographic primitive has on the environment in which it executes. Such effects include the power the device consumes [18, 19], its electromagnetic emissions [10, 32], timing [1, 5, 29], micro-architectural components [11, 23], and even acoustic and photonic emanations [13, 20]. By measuring these effects, an attacker can obtain information on the internal state of the cryptographic algorithm, which can lead to compromising the security of the primitive.

In many cases, there is a gap between the information obtained through the side channel and secret information, such as plaintexts or keys, which the attacker may wish to recover [8]. Techniques to bridge this gap have been developed for multiple cryptographic schemes [3, 6, 7, 16, 24, 25].

For RSA [33], in many cases the side-channel information provides the private key directly, requiring no further analysis [18, 31, 35]. When only partial information on the private key is available, there are two main approaches for key recovery. The Coppersmith method factors the RSA public modulus $N = pq$ given enough consecutive bits of the private prime p [7]. The Heninger-Shacham (HS) algorithm [16] exploits algebraic relationships among the two private primes p

and q, the private exponent d, and the two partial private exponents d_p and d_q, which are used in some implementations of RSA. Past works have used the HS algorithm to correct errors when the attacker obtains a degraded version of the key [15, 16, 30], to correct errors in side-channel information [15, 17, 21, 26, 27, 28, 30], and to recover information that is not obtained through the side channel [2, 4, 36].

In this work we consider the case that an attacker obtains knowledge of *digit equivalence* of the partial private exponents d_p and d_q. Specifically, we assume that the exponents are represented as digits in radix 2^ω and that the attacker can find which digits of the representation are the same without knowing the values of the digits themselves. Past works showed that such information can be obtained through side-channel attacks on fixed-window implementations [12] and that similar information can be obtained for sliding window implementations [17, 22, 34].

A naive approach for recovering the key from the digit equivalence information is to brute force the values of each of the 2^ω digits. However, such approach requires testing $2^\omega!$ combinations, or an expected complexity of $2^\omega!/2$. This complexity requires significant resources even for $\omega = 4$ and is prohibitive for the commonly used case of $\omega = 5$. Past works overcome this limitation by relying on additional information from the precomputation stage of the fixed-window and sliding window algorithms. Since hardening modular exponentiation against side-channel attacks requires additional resources, it may be tempting to harden the precomputation stage and rely on the complexity of recovering the key from the digit equivalence for side-channel protection.

Our Contribution

In this work we show how to apply the HS algorithm to the problem of recovering RSA private keys given digit equivalence. Specifically, we show how to use guesses for low significant digits to prune the search space of the HS algorithm when processing higher significant digits.

To analyse the complexity of our algorithm, we develop a theoretical model based on Markov chains. We use the model to calculate the probability of success and the number of operations required to recover the RSA key. Using this model we show that for the case of $\omega = 4$, more than 99% of the keys can be broken with a search space of size 2^{25}, well within the means of modestly resourced adversaries. For the common case of $\omega = 5$, the model predicts that 65% of the keys can be broken with a search space of 2^{40}, which is within the means of well resources adversaries.

We complement the theoretical analysis with concrete experiments, applying our algorithm to randomly generated RSA-2048 keys. We find that the model is highly accurate, correctly predicting the success and complexity of the attack. Specifically, for the case of $\omega = 4$, we can break 987 out of the 1000 keys we experiment, with a search space of 2^{25}.

2 Background

2.1 RSA

RSA [33] is a public key system that can be used for encryptions and for digital signatures. To generate an RSA key, Alice picks two random primes, p and q. The public key is (N, e), where $N = pq$, and e is chosen such that it is co-prime with $\varphi(N) = (p-1)(q-1)$. The private key is (p, q, d) where $d = e^{-1} \bmod \varphi(N)$. Most modern implementations use $e = 65537 = 2^{16} + 1$, and choose p and q to match the requirement.

We use $n = \lfloor \log_2 N \rfloor + 1$ to denote the bit length of the public modulus N. We further assume that the bit length of p and q is $n/2$.

To encrypt a message m, Bob calculates $c = m^e \bmod N$. To decrypt, Alice calculates $m = c^d \bmod N$. Signing a message m is done by calculating $s = m^d \bmod N$, and the signature is verified by testing that $m = s^e \bmod N$.

CRT-RSA. Alice can reduce the complexity of the private key operations using the Chinese Remainder Theorem (CRT). Specifically, Alice precomputes the CRT-RSA private key $(p, q, d, d_p, d_q, q_{inv})$, where $d_p = d \bmod (p-1)$, $d_q = d \bmod (q-1)$, and $q_{inv} = q^{-1} \bmod p$. To calculate $c^d \bmod N$, Alice then computes:

$$\begin{aligned} m_p &= m^{d_p} \bmod p \\ m_q &= m^{d_q} \bmod q \\ h &= q_{inv}(m_p - m_q) \bmod p \\ m &= m_q + hq. \end{aligned}$$

2.2 Fixed-Window Exponentiation

The fixed-window exponentiation algorithm, shown in Algorithm 1, calculates $B^E \bmod M$. The algorithm, parameterised by a window size ω, represents the exponent E as a number in radix 2^ω. We use the notation $E[\![i]\!]$ to refer to the ith digit of E. That is, E is represented as a sequence of digits $0 \le E[\![i]\!] < 2^\omega$, such that $E = \sum E[\![i]\!] 2^{\omega i}$.

To perform the exponentiation, the algorithm first precomputes 2^ω values $B_i = B^i \bmod M$. It then initialises an intermediate result r to 1 and proceeds to scan the exponent E digit by digit from the most significant to the least significant. For each digit $E[\![i]\!]$, the algorithm raises r to the power of 2^ω modulo M using squaring ω times, each time reducing the result modulo M. It then multiplies the result by the precomputed value $B_{E[\![i]\!]} = B^{E[\![i]\!]} \bmod M$, again reducing modulo M. At the end of the algorithm we have $r = B^{\sum E[\![i]\!] 2^{\omega i}} \bmod M = B^E \bmod M$.

2.3 Attacks on Fixed-Window Exponentiation

While the fixed-window algorithm is fairly regular and does not use secret-dependent control flow, implementations may still leak information about the

Algorithm 1: Fixed-window exponentiation

```
input  : window size $\omega$, base $B$, modulo $M$,
         exponent $E = \sum E[\![i]\!]2^{\omega i}$ with $0 \leq E[\![i]\!] < 2^{\omega}$.
output: $B^E \bmod M$

//Precomputation
$B_0 \leftarrow 1$
for $j$ from $1$ to $2^{\omega} - 1$ do
    $B_j \leftarrow B_{j-1} \cdot B \bmod M$
end

//Exponentiation
$r \leftarrow 1$
for $i$ from $|E| - 1$ downto $0$ do
    for $j$ from $1$ to $\omega$ do
        $r \leftarrow r^2 \bmod M$
    end
    $r \leftarrow r \cdot B_{E[\![i]\!]} \bmod M$
end
return $r$
```

digit being processed in each iteration. In some cases, the attacker can recover (some of) the bits of each digit $E[\![i]\!]$ [36]. However, a common leakage identifies digit equivalence, i.e. detecting when two digits $E[\![i]\!]$ and $E[\![j]\!]$ are the same without identifying the digits themselves [12, 34]. Specifically, Genkin et al. [12] uses a cache attack [22] to detect victim access patterns to the same digit, and Walter [34] exploits a differential power analysis [19] to identify repeating patterns in power traces. Several works recover similar information from sliding window implementations of modular exponentiation [17, 22]. All these works exploit leakage during the precomputation phase to recover the key. Specifically, when computing B_{i+1}, Algorithm 1 uses B_i. The order of precomputation is known, thus an attacker that identifies the use of B_i in the exponentiation phase can tie to its use in the precomputation phase and recover the digit value.

2.4 The Heninger-Shacham Algorithm

The Heninger-Shacham (HS) algorithm [16] uses a branch-and-prune approach for recovering an RSA private key from partial information on the bits of the components of the private key. Specifically, let (N, e) be an RSA public key and (p, q, d, d_p, d_q) be components of the corresponding private key, such that $N = pq$ is an n-bit RSA modulus with p and q primes, $e = 2^{16}+1$ is the public exponent, $d = e^{-1} \bmod (p-1)(q-1)$ is the private exponent, and $d_p = d \bmod p - 1$, $d_q = d \bmod q - 1$ are the CRT-RSA private exponents.

As Heninger and Shacham [16] note, there exist k, k_p, and k_q with $0 < k < e$ such that

$$\begin{aligned} N &= pq \\ ed &= k(N - p - q + 1) + 1 \\ ed_p &= k_p(p - 1) + 1 \\ ed_q &= k_q(q - 1) + 1. \end{aligned}$$

Moreover, Inci et al. [17] show that $0 < k_p, k_q < e$ and that given k_p we can find k_q and vice versa.

Let $\tau(x)$ be the exponent of the largest power of two that divides x. We note that because e is odd, $\tau(ed) = \tau(d)$, $\tau(ed_p) = \tau(d_p)$, and $\tau(ed_q) = \tau(d_q)$. Heninger and Shacham [16] first show how to find $d \bmod 2^{\tau(k)+2}$, $d_p \bmod 2^{\tau(k_p)+1}$, and $d_q \bmod 2^{\tau(k_q)+1}$. They then define a *slice* of the private key as

$$\mathsf{slice}(i) = (p[i], q[i], d[i + \tau(k)], d_p[i + \tau(k_p)], d_q[i + \tau(k_q)])\,.$$

where i indicates the bit index starting from the least significant bit. Therefore $p[i]$ is the ith bit of p and $p[0]$ refers to the least significant bit of p. Finally, they show that if we have a partial solution (p', q', d', d'_p, d'_q) for $\mathsf{slice}(0)$ to $\mathsf{slice}(i-1)$, the following four congruences hold.

$$p[i] + q[i] = (N - p'q')[i] \pmod 2 \tag{1}$$

$$d[i + \tau(k)] + p[i] + q[i] = (k(N + 1) + 1 - k(p' + q') - ed')[i + \tau(k)] \pmod 2 \tag{2}$$

$$d_p[i + \tau(k_p)] + p[i] = (k_p(p' - 1) + 1 - ed'_p)[i + \tau(k_p)] \pmod 2 \tag{3}$$

$$d_q[i + \tau(k_q)] + q[i] = (k_q(q' - 1) + 1 - ed'_q)[i + \tau(k_q)] \pmod 2 \tag{4}$$

Note that because p and q are primes and by the definition of $\tau(\cdot)$, we have that $\mathsf{slice}(0) = (1, 1, 1, 1, 1)$.

The HS algorithm has been proposed in the context of cold boot attacks [14], where most of the errors are that bits containing 1 may decay into 0. Further work has investigated the HS algorithm with unbalanced bidirectional errors [15, 30]. The HS algorithm has been further applied in the context of side-channel attacks which can have noisy measurements [21, 26, 27, 28]. The HS algorithm can be used to complete partial information obtained through cache attacks [2, 4, 36]

2.5 Markov Chains

This section introduces terminologies and relevant facts that we use in our analysis. We start with the definition of a Markov chain.

Definition 1. *A discrete-time stochastic process $\{X_n\}_{n\in\mathbb{Z}^+}$ on a countable state space Ω is called a Markov chain if for every n*

$$Pr(X_n = x_n | X_{n-1} = x_{n-1}, \ldots, X_1 = x_1) = Pr(X_n = x_n | X_{n-1} = x_{n-1}).$$

The intuition for the definition is that the history of a Markov chain is only considered through the current state, and the knowledge of the previous states has no impact on the movement to the next state.

An additional property that we use in the modelling is *time-homogeneity*. This refers to the fact that the probabilities of transitioning to the next state, with knowledge of the current state, do no change over time. That is,

$$Pr(X_{n+1} = j | X_n = i) = Pr(X_n = j | X_{n-1} = i).$$

Using the property of time-homogeneity, we then define the probabilities of transitioning between two states as

$$p_{ij} = Pr(X_n = j | X_{n-1} = i).$$

These can be generalised to the k-step transition probabilities by considering the probability of transitioning between states in k steps. That is,

$$p_{ij}^k = Pr(X_{n+k} = j | X_n = i).$$

The analysis of Markov chains is greatly simplified knowing these properties. For example, we create a matrix P of the transition probabilities of each of the possible transitions where each entry of the matrix $[P]_{i,j} = p_{ij}$. This forms a stochastic matrix, where each row sums to 1, since each state has its own probability distribution. The advantage of creating this matrix is that we can easily compute the k-step transition probabilities by taking powers of the matrix P [9, Theorem 1.1]. Then we have that the k-step transition probabilities can be calculated as

$$p_{ij}^k = [P^k]_{i,j}.$$

Therefore, describing a problem in this way enables us to convert a potentially computationally difficult problem of calculating the probabilities of moving between two states in k steps, from a combinatorial problem, whose complexity grows quickly in the number of steps, to a linear algebra problem of taking matrix powers.

3 Attacker Model

Recall that the aim is to recover the secret exponent E, where E represents d_p and d_q, used during the RSA exponentiation routine. The attacker knows that the victim performs the exponentiation using a fixed-window method whose width ω is publicly known.

We assume that the attacker can observe, via the side channel, the digit equivalence of the secret exponent. Note that the attacker does not know the values of those digits; the attacker only knows whether, for example, $E[\![i]\!]$ equals $E[\![i]\!]$ for $i \neq j$.

Figure 1 illustrates digit equivalence for $\omega = 4$. The attacker *does know* that $E[\![2]\!] = E[\![4]\!] = E[\![7]\!]$ but *does not know* that they are 1010. Similarly, the

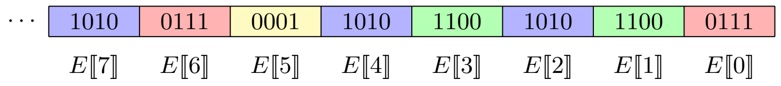

Fig. 1: A visualisation of digit equivalence for exponent E with $\omega = 4$.

attacker knows that $E[\![0]\!] = E[\![6]\!]$ but does not know their value. Moreover, the attacker also knows that $E[\![0]\!] \neq E[\![1]\!] \neq E[\![2]\!] \neq E[\![5]\!]$.

Continuing with this example, $\omega = 4$ has $2^\omega = 2^4 = 16$ possible different values of the digits. This means that the naive approach of determining the digits requires $16! \approx 2^{44}$. With well funded organisations, this attack is feasible. However, commonly used ω is usually larger than this.

Consider $\omega = 5$ as used in OpenSSL [36], there are a total of $2^5 = 32$ different digit values. The naive approach would require $32! \approx 2^{118}$, rendering this attack infeasible, even for well funded organisations such as the NSA.

4 Our Approach

We apply the HS algorithm with a branch-and-bound strategy together with pruning from our knowledge of digit equivalence. Recall that the HS algorithm reconstructs the CRT-RSA private components by looking at slices of the private key (p, q, d, d_p, d_q). At every $\mathsf{slice}(i)$, the algorithm builds the key by satisfying the four congruence relations described in Equations (1) to (4).

4.1 Algorithm Overview

In addition to the HS algorithm, we take advantage of side-channel information regarding digit equivalence. This allows us to further prune the solution space by removing any solutions that do not agree with our knowledge of digit equivalence. As a consequence, we significantly reduce the solution space and can reconstruct the key for larger window widths.

Our algorithm follows the HS algorithm and starts building the solution space from the least significant bit denoted by bit 0. When considering d_p and d_q (hence k_p and k_q), recall that $\mathsf{slice}(i)$ considers the bits $d_p[i + \tau(k_p)]$ and $d_q[i + \tau(k_q)]$. This results in two scenarios. One is where $\tau(k_p)$ and $\tau(k_q)$ are zero, which we denote as the *aligned case*. The other one is where $\tau(k_p)$ and $\tau(k_q)$ are not zero, which we denote as the *unaligned case*. We begin our analysis with the aligned case. The unaligned case is discussed in Section 4.4

4.2 Complexity Analysis of the Aligned Case

Assume the RSA fixed-window exponentiation uses a window width ω. This means that there are 2^ω different digits. Recall that we consider $\mathsf{slice}(i)$ and build bit i for p, q, d, d_p and d_q. Furthermore, recall that $\mathsf{slice}(0)$ is known. Because we assume $\tau(k_p) = \tau(k_q) = 0$, we know exactly one bit of the first (least significant) digit of each of d_p and d_q. Consequently, for the first digit, there remain $2^{\omega-1}$ possible partial solutions. Because we do not know any further information about the first digit, we cannot prune any possible solution at this step.

In the subsequent slices, there are two possibilities for pruning.

1. The first case is where the value of the current digit of d_p (or d_q) is equivalent to one of the previously observed digits. We can compare the current partial solution at each bit slice of the current digit and reject the solutions that do not match the bits of the equivalent digit seen previously.
2. The second case is where the value of the current digit is not equivalent to any of the previously observed digits. In this case, we can eliminate solutions where the value of its current digit equals a value of a previously seen digit.

In our algorithm, we model the search space as a search tree. The starting partial solution at $\mathsf{slice}(0)$ is at the root. Each level of the tree is a slice of the partial solution. The tree width reflects the number of solutions kept after pruning that level. For the purpose of the statistical analysis we make two assumptions about the statistical distributions of digits in the keys.

Assumption 1. (digit independence)
No correlation between lower and higher significant key bits. That is, given the knowledge of lower significant bits observed in the past, we do not gain further information regarding higher significant bits to be explored in the future.

Assumption 2. (key independence)
No dependency between p and q, thus d_p and d_q. This means that the knowledge of d_p (resp. d_q) does not provide additional information to infer d_q (resp. d_p).

We note that neither assumption hold in practice—Coppersmith [7] likely implies that Assumption 1 is invalid and Heninger and Shacham [16] invalidates Assumption 2. Hence, we only use them to facilitate the statistical analysis. The agreement between our model and the experiments indicates that violations of the assumptions do not result in significant statistical differences. We further note that any violation of these assumptions is likely to facilitate attacks on RSA.

We now consider $\mathsf{slice}(i)$. For the aligned case, $\mathsf{slice}(i)$ contains bits $d_p[i]$ and $d_q[i]$, which fall in digits $d_p[\![\lfloor i/\omega \rfloor]\!]$ and $d_q[\![\lfloor i/\omega \rfloor]\!]$, respectively. The side-channel information regarding the digit equivalence of $d_p[\![\lfloor i/\omega \rfloor]\!]$ and $d_q[\![\lfloor i/\omega \rfloor]\!]$ is categorised into four possibilities

(P1) Both $d_p[\![\lfloor i/\omega \rfloor]\!]$ and $d_q[\![\lfloor i/\omega \rfloor]\!]$ have been seen;
(P2) $d_p[\![\lfloor i/\omega \rfloor]\!]$ has been seen, but $d_q[\![\lfloor i/\omega \rfloor]\!]$ has not;
(P3) $d_q[\![\lfloor i/\omega \rfloor]\!]$ has been seen, but $d_p[\![\lfloor i/\omega \rfloor]\!]$ has not;
(P4) Neither $d_p[\![\lfloor i/\omega \rfloor]\!]$ nor $d_q[\![\lfloor i/\omega \rfloor]\!]$ has been seen.

Figure 2 illustrates seen and unseen digits of d_p and d_q, in the aligned case, where $\omega = 4$. Each box represents a digit whose value is printed within the box along with colors used to represent its value. The bit positions are given below

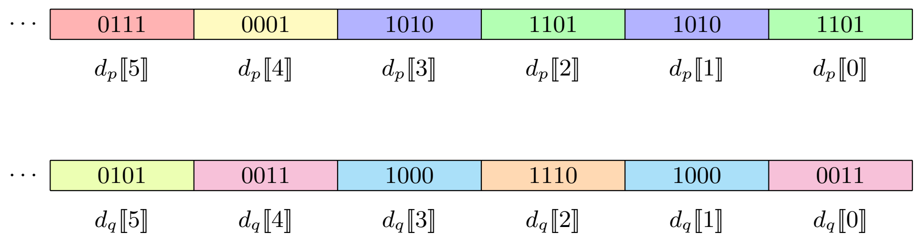
0111
0001
1010
1101
1010
1101
$d_p[\![5]\!]$ $d_p[\![4]\!]$ $d_p[\![3]\!]$ $d_p[\![2]\!]$ $d_p[\![1]\!]$ $d_p[\![0]\!]$
0101
0011
1000
1110
1000
0011
$d_q[\![5]\!]$ $d_q[\![4]\!]$ $d_q[\![3]\!]$ $d_q[\![2]\!]$ $d_q[\![1]\!]$ $d_q[\![0]\!]$

Fig. 2: An example of seen and unseen digits in the aligned case, $\omega = 4$.

the boxes. Recall that the attacker does not know the values of the digits; they only know the digit equivalence. Considering this scenario, where $\omega = 4$, there would be 2^3 solutions at the end of the first digit at $\mathsf{slice}(3)$. The subsequent digits for d_p and d_q are both unseen, corresponding to **(P4)**, so pruning can only occur at the end of the digit at $\mathsf{slice}(7)$. Using the notation introduced previously, any solution where $E[\![1]\!] = E[\![0]\!]$ in either d_p or d_q are pruned. Moving on to the next digit, we get scenario **(P2)**. The digit $d_p[\![2]\!]$ has been seen previously in $d_p[\![0]\!]$ and thus pruning could occur at each $\mathsf{slice}(8)$ to $\mathsf{slice}(11)$, i.e. solutions where $E[\![2]\!] \neq E[\![0]\!]$ for d_p are pruned. Additional pruning could occur from the unseen digit of $d_q[\![2]\!]$ at $\mathsf{slice}(11)$. The search continues on and the pruning at each slice depends on whether the current digit has been seen.

Because the side-channel information only applies to full digits, i.e. groups of ω bits, we only perform the pruning at a digit boundary. That is, we combine ω steps of the HS algorithm. To simplify notation, we use γ to refer to the digit number where bit i falls, i.e. $\gamma = \lfloor i/\omega \rfloor$.

Let y_γ and z_γ be the numbers of unique digits that have been observed at $d_p[\![0]\!], \ldots, d_p[\![\gamma]\!]$ and $d_q[\![0]\!], \ldots, d_q[\![\gamma]\!]$, respectively. We now make the concept of a previously seen digit more concrete by saying that a digit $d_p[\![\gamma]\!]$ (resp. $d_q[\![\gamma]\!]$) has been seen before if $y_\gamma = y_{\gamma-1}$ (resp. $z_\gamma = z_{\gamma-1}$).

Define two random variables Y_γ and Z_γ from the space $\{0, \ldots, 2^\omega - 1\}$ for the number of *unique* digits observed after reading γ digits. Hence, the four possibilities above, i.e., **(P1)**–**(P4)**, correspond to the four possibilities in Equation 5 for moving to observe the next slice. That is, given the previous value $(y_{\gamma-1}, z_{\gamma-1})$, we obtain the following probabilities:

$$Pr\left((Y_\gamma, Z_\gamma) = (y_\gamma, z_\gamma) | (Y_{\gamma-1}, Z_{\gamma-1}) = (y_{\gamma-1}, z_{\gamma-1})\right)$$
$$= \begin{cases} \left(\frac{y_{\gamma-1}}{2^\omega}\right)\left(\frac{z_{\gamma-1}}{2^\omega}\right) & \text{if } y_\gamma = y_{\gamma-1} \text{ and } z_\gamma = z_{\gamma-1} \\ \left(\frac{y_{\gamma-1}}{2^\omega}\right)\left(\frac{2^\omega - z_{\gamma-1}}{2^\omega}\right) & \text{if } y_\gamma = y_{\gamma-1} \text{ and } z_\gamma = z_{\gamma-1}+1 \\ \left(\frac{2^\omega - y_{\gamma-1}}{2^\omega}\right)\left(\frac{z_{\gamma-1}}{2^\omega}\right) & \text{if } y_\gamma = y_{\gamma-1}+1 \text{ and } z_\gamma = z_{\gamma-1} \\ \left(\frac{2^\omega - y_{\gamma-1}}{2^\omega}\right)\left(\frac{2^\omega - z_{\gamma-1}}{2^\omega}\right) & \text{if } y_\gamma = y_{\gamma-1}+1 \text{ and } z_\gamma = z_{\gamma-1}+1 \end{cases} \tag{5}$$

We derive these probabilities by utilising the independence of the two keys, d_p and d_q. Therefore, we consider each contribution to the probability separately and the contribution is either the proportion of seen digits or the proportion of unseen digits, depending on whether the current digit has been seen in the key stream. Therefore, we can calculate the probability of a particular key sequence by multiplying the probability of the individual components. In Section 4.3 we discuss the formulation of these probabilities into two independent Markov chains to model the key recovery.

Note that the complexity of our attack depends on the size of the search space, i.e. the number of nodes in the search tree. Let W_γ be a random variable that denotes the search space (the number of possible candidate keys) or tree width after γ digit steps. The change of the width at each step is defined as follows.

$$\begin{cases} \frac{1}{2^\omega} & \text{if } y_\gamma = y_{\gamma-1} \text{ and } z_\gamma = z_{\gamma-1} \\ \frac{2^\omega - z_\gamma}{2^\omega} & \text{if } y_\gamma = y_{\gamma-1} \text{ and } z_\gamma = z_{\gamma-1}+1 \\ \frac{2^\omega - y_\gamma}{2^\omega} & \text{if } y_\gamma = y_{\gamma-1}+1 \text{ and } z_\gamma = z_{\gamma-1} \\ \frac{(2^\omega - y_\gamma)(2^\omega - z_\gamma)}{2^\omega} & \text{if } y_\gamma = y_{\gamma-1}+1 \text{ and } z_\gamma = z_{\gamma-1}+1 \end{cases}$$

Observe that the change in the width only depends on the number of unique digits that have been seen and the number of digits scanned, γ. In other words, the width *is not* dependent upon the sequence of Y_γ or Z_γ but the value at γ. Since we know that the first digit must be odd (due to being prime), the first bit must be one. This means that there are fewer possibilities for the first digit. Consequently, this gives a factor of $2^{\omega-1}$ for the first width since we have one fewer binary choice for the first bit. Therefore, the width after reading in γ digits from each of d_p and d_q is

$$\frac{2^{\omega-1}}{2^{\gamma\omega}} \prod_{m=1}^{y_\gamma} (2^\omega - m) \prod_{m=1}^{z_\gamma} (2^\omega - m) \,. \tag{6}$$

As noted previously, the width is independent of the order of the sequence. The expression takes the product of the y_γ and z_γ numerators of the change in widths and the initial width $2^{\omega-1}$, then divides by the number of 2^ω for γ digits scanned. Assume the threshold of 2^t, we have

$$\frac{2^{\omega-1}}{2^{\gamma\omega}} \prod_{m=1}^{y_\gamma} (2^\omega - m) \prod_{m=1}^{z_\gamma} (2^\omega - m) \geq 2^t$$

and therefore to exceed the threshold we need

$$\prod_{m=0}^{y_\gamma} (2^\omega - m) \prod_{m=0}^{z_\gamma} (2^\omega - m) \geq 2^{t-1+(\gamma-1)\omega}.$$

4.3 Independent Markov Chains

Recall the two assumptions in our analysis, namely, digit independence (i.e. previously observed digits do not determine unexplored digits) and key independence (i.e. knowing d_p does not infer d_q or vice versa). We use these properties to create identical distributed Markov chains and analyse these chains operating on d_p and d_q independently. Each Markov chain has the state space $\Omega = \{0, \ldots, 2^\omega - 1\}$ whose transitions have two possibilities:

1. Sample a digit that has previously been seen, or
2. Sample a digit that has not been seen.

Therefore, we define the probability transitions as

$$y_\gamma \to \begin{cases} y_{\gamma-1} & \text{with probability } \dfrac{y_{\gamma-1}}{2^\omega} \\ y_{\gamma-1} + 1 & \text{with probability } \dfrac{2^\omega - y_{\gamma-1}}{2^\omega}. \end{cases}$$

Using this formulation, we construct a probability transition matrix P. An example for $\omega = 4$ state chain is given below. Notice that the matrix has non-zero probabilities on the main diagonal and the diagonal above only.

$$P = \begin{bmatrix} \frac{1}{4} & \frac{3}{4} & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{3}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let Y_γ be a random variable for the number of unique digits observed at digit γ. Thus, the Markov chain tracking the evolution of either d_p or d_q is

$$\mathcal{I} = \{Y_\gamma\}_{\gamma \in \{0,\ldots,2^\omega-1\}}.$$

Given the Markov chain structure, we can calculate the probabilities of having observed y_γ unique digits, after observing γ digits.

Note that this Markov chain is time-homogeneous since the transition probabilities do not change with the number of digits that have been read. This allows the calculation of the probabilities at each number of digit γ read as the γth power of the probability transition matrix. That is,

$$Pr(Y_\gamma = y_\gamma | Y_1 = y_1) = [P^k]_{y_1,y_\gamma}$$

since both d_p and d_q are independent and identically distributed.

The probability transition matrix P and its powers completely determine the system. Note also that we consider the initial state of beginning the first digit with a single unique value. This means that we will always consider the top row of the matrix P for calculation of the relevant probabilities,

To finalise this discussion, thanks to the independence assumptions, we can calculate the transition probabilities for random variables Y_γ and Z_γ, the number of unique digits observed by digit γ of d_p and d_q respectively,

$$Pr(Y_\gamma = l, Z_\gamma = m) = Pr(Y_\gamma = l)Pr(Z_\gamma = m) = [P^\gamma]_{1,y_\gamma}[P^\gamma]_{1,z_\gamma}.$$

Note that the index of the starting state is 1, as we consider the probability of moving from state 1 to j in γ steps.

Regarding computational complexity, the naive calculation seen previously required that at each step, ω probabilities are calculated, thus resulting in the complexity $O(\omega^\gamma)$. Utilising the Markov chain approach, the calculation is reduced to taking the γ powers of P where each matrix multiplication is $O((2^\omega)^{2.37}) = O(2^{2.37\omega})$. This means that our approach has the computational complexity of $O(\gamma 2^{2.37\omega})$.

4.4 Unaligned Case

As previously mentioned, many real examples do not begin scanning digits from the least significant bit, i.e. $\tau(k_p)$ and $\tau(k_q)$ are not zero. Our analysis suggests modelling these offsets as independent geometric random variables. That is, each bit stream of d_p and d_q, the offset O has probability of occurring of

$$Pr\left(O = o\right) = \frac{1}{2^{o+1}}, o \in \{0, 1, \ldots\}.$$

We calculate the change in width by taking the assumption that all bits before the offset are known and we retain knowledge of their values. This has no impact on the evolution of the Markov chain. Therefore, we can utilise the same probability transitions while adjusting the weight by the offset. As a result, given the offsets o_p and o_q, we have that the width can be calculated as

$$\left(\frac{1}{2^{o_p}}\right)\left(\frac{1}{2^{o_q}}\right)\left(\frac{2^{\omega-1}}{2^{k\omega}}\right)\prod_{m=1}^{y_\gamma}\left(2^\omega - m\right)\prod_{m=1}^{z_\gamma}\left(2^\omega - m\right). \tag{7}$$

That is, we derive this expression from Equation 6 and adjusting for the known bits, which are those before the offset. Note that this means that any offset provided will lower the width for the same digit γ, and number of digits observed, y_γ and z_γ. Therefore, the case with no offsets for scanning digits will have the largest width, for identical keys.

We calculate the expected width at each digit γ by summing over widths of the offset Equation 7 and the probabilities of being in a state (y_γ, z_γ) from Equation 5. Explicitly, this gives the expected width for a digit γ of

$$E[W_\gamma]=\sum_{y_\gamma\leq\gamma}\sum_{z_\gamma\leq\gamma}\sum_{o_p}\sum_{o_q}[P^\gamma]_{1,y_\gamma}[P^\gamma]_{1,z_\gamma}\frac{1}{2^{o_p}}\cdot\frac{1}{2^{o_q}}\cdot\frac{2^{\omega-1}}{2^{\gamma\omega}}\prod_{m=1}^{y_\gamma}\left(2^\omega-m\right)\prod_{m=1}^{z_\gamma}\left(2^\omega-m\right).$$

5 Results and Comparisons

We theoretically and experimentally evaluate our approach of reconstructing RSA private keys given side-channel information of digit equivalence. For the former, we use our derived formulas to estimate the search space and success probability. For the latter, we run our algorithm on a high-performance cluster and observe the convergence to the solution. In both cases, we also set threshold on the space complexity.

5.1 Theoretical Results

For the theoretical evaluation, we consider the RSA fixed-window exponentiation with $\omega = 4, 5$ and 6. For each ω, we use three different thresholds corresponding to three computation budgets. These three thresholds are $2^{25}, 2^{40}$ and 2^{60} which represent resource-constrained attackers, well-funded organisations, and nation-state organisations such as NSA.

The results for $\omega = 4$ are shown in Figure 3. This suggests that the algorithm can recover the majority of the key before reaching the lowest resource-constrained attackers threshold of 2^{25}. To be more precise, 99.9% of the key can successfully be recovered. Further analysis shows that the keys that exceed the maximum width have many consecutive unique digits at the beginning of the key. Thus, this allows the width to grow much quicker than the ability to prune infeasible keys.

The results for $\omega = 5$ is shown in Figure 4. As expected, the percentiles, median and mean all increase as the window width increases from $\omega = 4$ to $\omega = 5$. Even though it becomes more challenging for resource-constrained attackers, it is feasible for well-funded organisations.

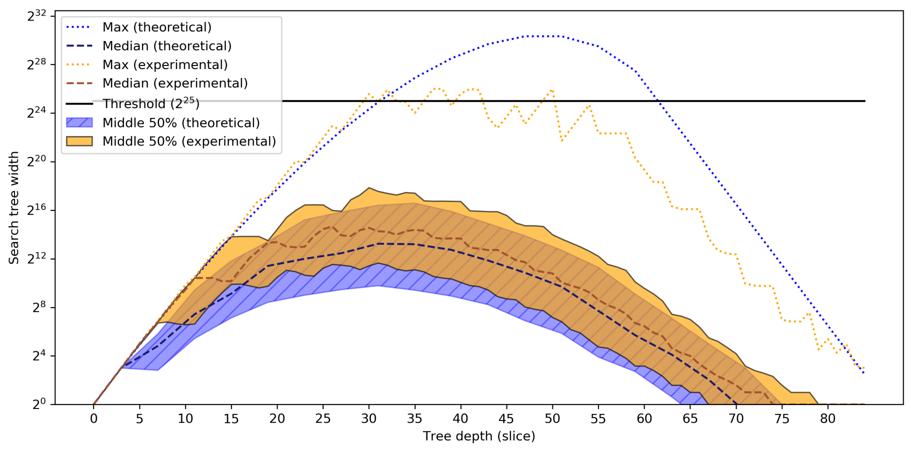

Fig. 3: Distribution of width seen at each slice for $\omega = 4$ for both the theoretical and experimental results.

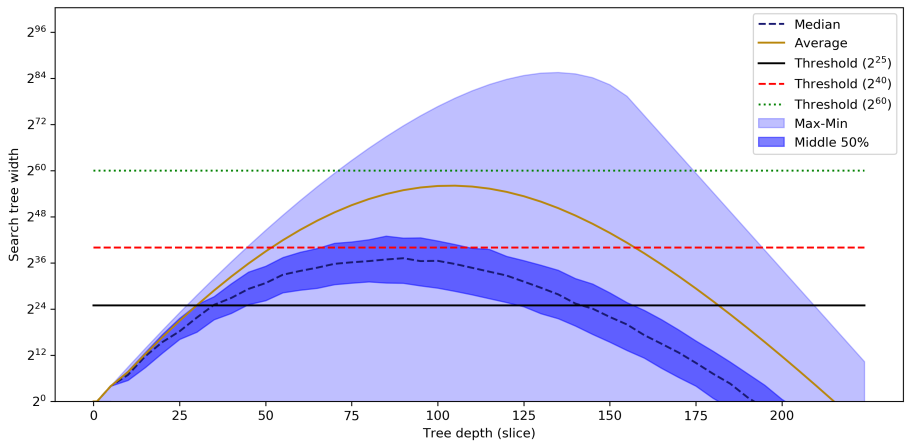

Fig. 4: Distribution of width seen at each slice of the Markov chain model, $\omega = 5$.

These plots give insight into how the key recovery behaves. Most of the typical behaviour of keys is contained within a relatively small band demonstrated by the middle 50%. The mean is higher than the 75th percentile, which highlights that the maxima tend to be much higher than the middle values. The influence of the unaligned keys lowers the width in general, as the resulting reduction in width is a power-of-two offset. This has a large impact in lowering the median and percentile ranges, since many combinations of unaligned keys still occur with high probability, while have large reductions to the size of the width. Table 1 summarises the success probability and the threshold for window width $\omega = 4, 5$ and 6.

Table 1: Success rate in reconstructing keys for different ω and thresholds

	2^{25}	2^{40}	2^{60}
$\omega = 4$	99.9%	100.0%	100.0%
$\omega = 5$	8.2%	64.8%	99.9%

5.2 Experimental Results

To demonstrate the practicality of our attack, we implement and run the attack on a high-performance cluster. We are interested in the behaviour of convergence to a solution before reaching the search tree width threshold which we set to 2^{25} (resource-constrained attackers). If the solution space exceeds this threshold width, the search is abandoned as it would be too computationally intensive to continue the search.

The distribution of the search tree width at each bit slice for $\omega = 3$ and 4 are shown in Figure 5 and Figure 3 respectively. It shows the widths up to $\mathsf{slice}(85)$. The widths at the subsequent layers fluctuate between 1 and 2. One of these solution is the key in which we want to recover. These results are generated with 1000 samples with randomly generated secret values. The key is randomly generated with $e = 65537$ and with 2048-bit RSA modulus.

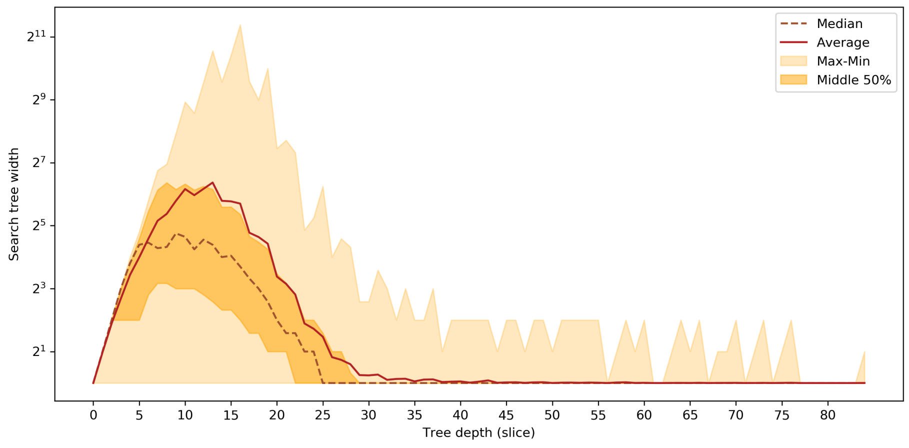

Fig. 5: Distribution of width seen experimentally at each slice, $\omega = 3$.

The success rate for $\omega = 4$ is 98.7%. From these results, we can observe that the search widths follow a general pattern of an exponential-like increase before an exponential-like decrease. This aligns with our understanding. Initially, building the key with no seen digits, we expect there to be a growing number of possibilities for the digit string. As we move further along the slices of the key, at some point, we would have enough information of the key to narrow the search space.

Table 2 lists the values of the widths seen when $\omega = 4$ and threshold is 2^{25}. Note that the maximum and minimum width seen in the experimental results is not the true maximum and minimum as it is affected by the attacker-imposed threshold. The search is abandoned when the tree width exceeds the threshold, so we would get that the tree width drops to zero when an experiment passes the threshold, and the maximum width captured will be the width right before the run was abandoned. The theoretical results also accounts for extremely low probability events; thus a higher theoretical maximum is expected.

Table 2: Comparison of the width seen when $\omega = 4$ and threshold 2^{25}

	Theoretical		Experimental	
	bit index	maximum	bit index	maximum
mean	39	570 000	38	790 000
min*	3	8	20	3
25%	31	880	31	33 000
median	31	9 700	26	26 000
75%	35	98 000	30	240 000
max*	47	1 400 000 000	38	67 000 000

* Expect different theoretical and experimental results. The experimental results stop after the search tree exceeds the threshold.

Despite the differences of the success rate, 98.7% from experiments and 99.9% using the theoretical model, the general behaviour of the search tree is similar. We see this in Figure 3 and Table 2. Ignoring the minimum and maximum due to the difference in performing the search experimentally, the mean of the tree width reaches its peak around bit index 40 with roughly the same value in both theoretical and experimental results. The middle 50% range occurs roughly at bit index 32, although the ranges of this value is much lower in the theoretical results. Again, this could be due to the modelling accounting for extremely low probability events.

6 Conclusions

In this work we apply the Heninger-Shacham algorithm in a new context. We assume a side-channel adversary that can observe the equivalence of digits in the private exponents of CRT-RSA. We show how to apply the algorithm given such information and develop a theoretical model that allows us to analyse the complexity of the attack. The model shows that the attack is feasible for a suitably funded organisation with a window size of 5 bits. We further validate the model through experimentation with randomly chosen RSA keys.

Our model assumes that the digit equivalence information is complete. A potential extension of this work is to evaluate cases where we have partial information. For example, when there are errors in the digit equivalence information or when we only know the class of the digits (e.g. the Hamming weight). The work could also be extended to consider cases that use sliding window exponentiation.

The results presented here has, once again, made apparent the importance of using constant-time implementations against side-channel attacks.

Acknowledgements

We would like to thank all reviewers for providing insightful feedback, which has improved the paper.

This work was supported by an ARC Discovery Early Career Researcher Award (project number DE200101577); an ARC Discovery Project (project number DP210102670); The Blavatnik ICRC at Tel-Aviv University; the Phoenix HPC service at the University of Adelaide; and gifts from Google and Intel.

References

[1] Daniel J. Bernstein. *Cache-timing attacks on AES*. Preprint available at `http://cr.yp.to/papers.html#cachetiming`. 2005.

[2] Daniel J. Bernstein, Joachim Breitner, Daniel Genkin, Leon Groot Bruinderink, Nadia Heninger, Tanja Lange, Christine van Vredendaal, and Yuval Yarom. "Sliding Right into Disaster: Left-to-Right Sliding Windows Leak". In: *CHES*. 2017, pp. 555–576.

[3] Dan Boneh and Ramarathnam Venkatesan. "Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes". In: *CRYPTO*. 1996, pp. 129–142.

[4] Joachim Breitner. *More on sliding right*. Cryptology ePrint Archive 2018/1163. `http://eprint.iacr.org/2018/1163/`. 2018.

[5] Billy Bob Brumley and Nicola Tuveri. "Remote Timing Attacks Are Still Practical". In: *ESORICS*. 2011, pp. 355–371.

[6] Chitchanok Chuengsatiansup, Daniel Genkin, Yuval Yarom, and Zhiyuan Zhang. "Side-Channeling the Kalyna Key Expansion". In: *CT-RSA*. 2022.

[7] Don Coppersmith. "Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known". In: *EUROCRYPT*. Vol. 1070. 1996, pp. 178–189.

[8] Gabrielle De Micheli and Nadia Heninger. *Recovering cryptographic keys from partial information, by example*. Cryptology ePrint Archive, Report 2020/1506. `http://eprint.iacr.org/2020/1506/`. 2020.

[9] Richard Durrett and R Durrett. *Essentials of stochastic processes*. Vol. 1. Springer, 1999.

[10] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. "Electromagnetic Analysis: Concrete Results". In: *CHES*. 2001, pp. 251–261.

[11] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware". In: *J. Cryptographic Engineering* 8.1 (2018), pp. 1–27.

[12] Daniel Genkin, Lev Pachmanov, Eran Tromer, and Yuval Yarom. "Drive-By Key-Extraction Cache Attacks from Portable Code". In: *ACNS*. 2018, pp. 83–102.

[13] Daniel Genkin, Adi Shamir, and Eran Tromer. "RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis". In: *CRYPTO*. 2014, pp. 444–461.

[14] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. "Lest We Remember: Cold Boot Attacks on Encryption Keys". In: *USENIX Security*. 2008, pp. 45–60.

[15] Wilko Henecka, Alexander May, and Alexander Meurer. "Correcting Errors in RSA Private Keys". In: *CRYPTO*. 2010, pp. 351–369.

[16] Nadia Heninger and Hovav Shacham. "Reconstructing RSA Private Keys from Random Key Bits". In: *CRYPTO*. 2009, pp. 1–17.

[17] Mehmet Sinan Inci, Berk Gülmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. “Cache Attacks Enable Bulk Key Recovery on the Cloud”. In: *CHES*. 2016, pp. 368–388.

[18] Paul C. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. In: *CRYPTO*. 1996, pp. 104–113.

[19] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. “Differential Power Analysis”. In: *CRYPTO*. 1999, pp. 388–397.

[20] Juliane Krämer, Dmitry Nedospasov, Alexander Schlösser, and Jean-Pierre Seifert. “Differential Photonic Emission Analysis”. In: *COSADE*. 2013, pp. 1–16.

[21] Noboru Kunihiro and Junya Honda. “RSA Meets DPA: Recovering RSA Secret Keys from Noisy Analog Data”. In: *CHES*. 2014, pp. 261–278.

[22] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. “Last-Level Cache Side-Channel Attacks are Practical”. In: *IEEE SP*. 2015, pp. 605–622.

[23] Xiaoxuan Lou, Tianwei Zhang, Jun Jiang, and Yinqian Zhang. “A Survey of Microarchitectural Side-channel Vulnerabilities, Attacks and Defenses in Cryptography”. In: *CoRR* abs/2103.14244 (2021).

[24] Phong Q. Nguyen and Igor E. Shparlinski. “The Insecurity of the Digital Signature Algorithm with Partially Known Nonces”. In: *J. Cryptol.* 15.3 (2002), pp. 151–176.

[25] Phong Q. Nguyen and Igor E. Shparlinski. “The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces”. In: *Des. Codes Cryptogr.* 30.2 (2003), pp. 201–217.

[26] Kento Oonishi, Xiaoxuan Huang, and Noboru Kunihiro. “Improved CRT-RSA Secret Key Recovery Method from Sliding Window Leakage”. In: *ICISC*. 2019, pp. 278–296.

[27] Kento Oonishi and Noboru Kunihiro. “Attacking Noisy Secret CRT-RSA Exponents in Binary Method”. In: *ICISC*. 2018, pp. 37–54.

[28] Kento Oonishi and Noboru Kunihiro. “Recovering CRT-RSA Secret Keys from Noisy Square-and-Multiply Sequences in the Sliding Window Method”. In: *ACISP*. 2020, pp. 642–652.

[29] Dan Page. *Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel*. Cryptology ePrint Archive, Report 2002/169. http://eprint.iacr.org/2002/169/. 2002.

[30] Kenneth G. Paterson, Antigoni Polychroniadou, and Dale L. Sibborn. “A Coding-Theoretic Approach to Recovering Noisy RSA Keys”. In: *ASIACRYPT*. 2012, pp. 386–403.

[31] Colin Percival. “Cache Missing for Fun and Profit”. In: *BSDCan 2005*. 2005. URL: http://css.csail.mit.edu/6.858/2014/readings/ht-cache.pdf.

[32] Jean-Jacques Quisquater and David Samyde. “ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards”. In: *Smart Card Programming and Security*. 2001, pp. 200–210.

[33] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. In: *Commun. ACM* 21.2 (1978), pp. 120–126.

[34] Colin D. Walter. “Sliding Windows Succumbs to Big Mac Attack”. In: *CHES*. 2001, pp. 286–299.

[35] Yuval Yarom and Katrina Falkner. “Flush+Reload: A High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: *USENIX Security*. 2014, pp. 719–732.

[36] Yuval Yarom, Daniel Genkin, and Nadia Heninger. “CacheBleed: A Timing Attack on OpenSSL Constant Time RSA”. In: *CHES*. 2016, pp. 346–367.