How Secure Is Green IT? The Case of Software-Based Energy Side Channels

Heiko Mantel(✉), Johannes Schickel, Alexandra Weber(✉),
and Friedrich Weber

Department of Computer Science, TU Darmstadt, Darmstadt, Germany
{mantel,schickel,weber,fweber}@mais.informatik.tu-darmstadt.de

Abstract. Software-based energy measurement features in contemporary CPUs allow one to track and to limit energy consumption, e.g., for realizing green IT. The security implications of software-based energy measurement, however, are not well understood. In this article, we study such security implications of green IT. More concretely, we show that side-channel attacks can be established using software-based energy measurement at the example of a popular RSA implementation. Using distinguishing experiments, we identify a side-channel vulnerability that enables attackers to distinguish RSA keys by measuring energy consumption. We demonstrate that a surprisingly low number of sample measurements suffices to succeed in an attack with high probability. In contrast to traditional power side-channel attacks, no physical access to hardware is needed. This makes the vulnerabilities particularly serious.

1 Introduction

Controlling and limiting energy consumption is crucial for datacenters, both, ecologically and economically. Minimizing energy consumption is key to achieving both, green IT and higher datacenter density [17]. To support the achievement of energy-consumption goals, software-based energy measurement features have been introduced to CPUs by various vendors, e.g., by Intel [21, Chap. 14.9].

While the potential benefits of software-based energy measurement are clear [17], its security implications are not yet well-understood. To clarify such implications is our goal. More concretely, we focus on side channels that attackers might establish using software-based energy measurement. In a side-channel attack, an attacker extracts secrets, like cryptographic keys, from execution characteristics of a program, like running time [4,11,22], cache behavior [8,32,52], or power consumption [23,24,37]. Prior work on power-consumption side channels required specialized hardware or required the device under attack to use a battery.

In this article, we investigate the danger of side channels introduced by software-based energy measurement. We also evaluate the effectiveness of two candidate countermeasures against such side channels. To make things concrete, we focus on Intel RAPL, an energy measurement feature in Intel CPUs [21].

We perform our experiments on an Intel i5-4590 desktop CPU. In our experiments, we measure the energy consumption of a victim program purely in

J. Lopez et al. (Eds.): ESORICS 2018, LNCS 11098, pp. 218–239, 2018.
https://doi.org/10.1007/978-3-319-99073-6_11

software, using Intel RAPL. Based on our measurements, we evaluate qualitatively whether an attacker can learn secret information and then quantify this threat using statistical methods on a concrete decision procedure. Subsequently, we evaluate the effectiveness of countermeasures based on information theory.

Our main finding is that an attacker can distinguish between RSA secret keys purely by using software-based energy measurement. More concretely, the attacker can distinguish which secret key is used in the RSA implementation from the popular cryptographic library Bouncy Castle. We show that 7 observations suffice to guess the key correctly with a probability above 99%. This number of required observations is surprisingly low and the detected weakness in Bouncy Castle RSA is, hence, a serious concern. While it is clear that CPU features for increasing performance are common sources of side channels (see, e.g., caches [43] or branch prediction [1]), CPU features for controlling energy were not in the focus of research on side channels so far. Our results show that CPU features for controlling energy do introduce side channels and that these side channels are severe. That clarifies the security implications of green IT in this domain.

We investigate two candidate countermeasures against software-based energy side channels, namely the program transformations cross-copying [2] and conditional assignment [40]. We evaluate their effectiveness by the reduction in side-channel capacity that they achieve in our experiments. While cross-copying only reduces capacity by 8%, conditional assignment reduces capacity by 99%. Thus, conditional assignment could be a suitable basis for hardening security-critical implementations against software-based energy side channels.

In summary, our main contributions are (1) a qualitative and a quantitative analysis of software-based energy side channels at the example of Bouncy Castle RSA and Intel RAPL, and (2) a quantitative evaluation of the effectiveness of two candidate countermeasures against energy side channels.

2 Preliminaries

Side Channels. In 1996, Kocher showed that a naive square-and-multiply implementation of modular exponentiation is vulnerable to timing-side-channel attacks [22]. Modular exponentiation is, for example, used in RSA decryption to compute $p = c^d \pmod{n}$, for ciphertext c and secret key d [45].

```
 1: Input: $(d, n), c$
 2: $r \leftarrow 1$
 3: for $i = 1$ to $i = bitLength(d)$ do
 4:     if $d \,\%\, 2 == 1$ then
 5:         $r \leftarrow (r * y) \,\%\, n$
 6:     end if
 7:     $y \leftarrow (y * y) \,\%\, n$
 8:     $d \leftarrow d >> 1$
 9: end for
10: return $r \,\%\, n$
```

Fig. 1. Square&Multiply

A square-and-multiply implementation of modular exponentiation is given in Fig. 1. Line 5 is only executed when the condition in Line 4 evaluates to *true*. Execution of Line 5 takes additional time. Since the condition depends on bits from the exponent, the execution time of the program encodes the Hamming weight of the exponent. An attacker can exploit this variation in execution times to extract the secret exponent d [22].

In the style of Millen [39], a side channel can be modeled as an information-theoretic channel [15] with random variables X and Y as the input alphabet and output alphabet. The input alphabet are all secrets a program can process, and the output alphabet are all possible side-channel observations. The worst-case side-channel leakage can be measured by the channel capacity $C(X; Y)$ [15].

Software-Based Energy Measurement. Energy E (measured in J for *joule*) is the aggregation of instantaneous power consumption values $p(t)$ (measured in W for *watt*) over time, i.e., $E = \int_{t_0}^{t_1} p(t)dt$ [19]. Similar to [41], we define the energy consumption of a program as the energy consumed by the CPU and main memory during program execution (e.g., for arithmetics and accesses to data).

Running Average Power Limit (Intel RAPL) is a set of energy sensors on CPUs introduced with Intel's Sandy Bridge processor architecture [20]. While Intel RAPL's primary purpose is to enforce power consumption limits [21, Chap. 14], it also exposes the energy consumption of the CPU through the model-specific register (MSR) MSR_PKG_ENERGY_STATUS, which is updated every millisecond. The measurements provided are accurate [20]. Linux exposes Intel RAPL to userspace through the msr kernel module [31] and through the Power Capping framework (powercap) [30]. Both, msr and powercap, provide energy measurements in pseudo-files. The former can be accessed with root privileges, e.g., under /dev/cpu/0/msr for the first CPU. The latter can be accessed by non-privileged users under /sys/class/powercap/intel-rapl/intel-rapl:0/energy_uj. From powercap, energy measurements in the unit $\mu J = 10^{-6} J$ can be obtained.

Distinguishing Experiments. In a *distinguishing experiment*, two distinct secret inputs are passed to a program and a side channel output is repeatedly measured for each input. For instance, Mantel and Starostin use distinguishing experiments to show that a program exhibits a timing-side-channel vulnerability [36].

Based on the empirical data from a distinguishing experiment, statistical tools can be used to quantify the side-channel leakage of the program under test. For a given attacker strategy, the success probability can be computed based on hypothesis testing. Independent of an attacker strategy, the side-channel capacity $C(X; Y)$ of the program can be estimated with a statistical procedure (e.g., [12]).

A test of hypothesis is a tool to investigate conformance of a hypothesis H_0 with experimental data [48, p. 64]. We denote the alternative hypothesis by H_1. A test has two error cases: (a) the test wrongly accepts H_0 (a false positive), or (b) the test wrongly refutes H_0 (a false negative). The probabilities for a false positive and a false negative are denoted by $P(H_0|H_1)$ and $P(H_1|H_0)$.

The binomial distribution (or Bernoulli distribution) is the probability distribution for the number of successes in n independent experiments [48, p. 112]. The probability that in n experiments, each featuring success probability p, r successes are observed is $P_{n,p}(r) = \binom{r}{n} p^r p^{n-r}$, where $\binom{r}{n} = \frac{n!}{r!(n-r)!}$ is the binomial coefficient. We write $P_{n,p}(r \leq X) = \sum_{i=0}^{X} P_{n,p}(i)$ for the probability that at most X out of n experiments exhibit a success. Conversely, the probability that more than X out of n experiments exhibit a success is $P_{n,p}(r > X) = 1 - P_{n,p}(r \leq X)$.

Chothia and Smirnov show in [13] how tests of hypothesis can be used to attack e-passports. Based on a simple selection criterion, their distinguishing attack tests the hypothesis that the passport under attack belongs to the victim. Using $P(H_0|H_1)$ and $P(H_1|H_0)$, they calculate the number of observations needed to distinguish passports with error rates below 1%.

Program Transformations Against Side Channels. Multiple source-to-source program transformations were proposed for mitigating timing side channels, including cross-copying [2], conditional assignment [40], transactional branching [6], and unification [26]. The technique *cross-copying* pads branches by adding copies of the statements in one branch to the respective other branch. In the copies, dummy statements are used, which do not affect the program's state, but require the same execution time as the respective original statements. The technique *conditional assignment* removes secret-dependent branching completely and replaces assignments from the respective branches by assignments that are masked by the branching condition. Both, cross copying and conditional assignment were evaluated analytically and experimentally [2,36,40]. For instance, they were effective against the timing side channel in an implementation of Fig. 1 [36].

RSA in Bouncy Castle. Bouncy Castle is a cryptographic library for Java [29]. A provider class allows the use of Bouncy Castle through the Java Cryptography Extension (JCE). In the form of Spongy Castle [50], Bouncy Castle is widely used on Android, e.g., in the WhatsApp messenger [32]. Side channels in Bouncy Castle are, hence, a serious security threat. Recently, it was shown that Bouncy Castle 1.5's AES implementation is vulnerable to cache side-channel attacks [32].

Bouncy Castle contains implementations of various variants of the RSA asymmetric encryption scheme. The RSA encryption and decryption functionality is implemented in the Java class RSAEngine. RSAEngine can be used either directly or as backend in cipher modes, such as *OAEP* [7] and *PKCS1* [46]. An RSA key can be generated using the class RSAKeyPairGenerator.

3 Our Approach

In a side-channel attack, an attacker collects sample execution characteristics of a victim program. Based on these samples, the attacker distinguishes between the candidate secrets (e.g., valid crypto keys). The core of many side-channel attacks is to distinguish between candidate secrets from a restricted set (e.g., varying only in one bit [22] or byte [3,8]). For instance, AlFardan and Paterson [3] distinguish between two secret plaintexts based on the time that an implementation of TLS takes to decrypt them. Using distinguishing experiments [36], one can detect weaknesses in implementations that allow one to distinguish between secret inputs, e.g., as a basic step in a side-channel attack.

We define a general procedure for such experiments and use it to assess the implementation of RSA in Bouncy Castle with respect to two attacker models.

3.1 Procedure for Distinguishing Experiments

An implementation *imp* is assessed with respect to a particular security concern, namely the leakage of a secret input s to an attacker under an attacker

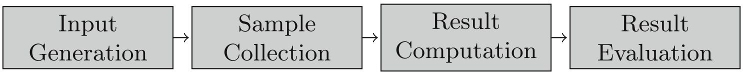

Fig. 2. Procedure for a distinguishing experiment

model a. For instance, *imp* could be an RSA implementation and s could be the secret RSA key. The assessment consists of four steps, visualized in Fig. 2: input generation, sample collection, result computation, and result evaluation.

In the first step, *input generation*, two input vectors to the implementation *imp* are generated, such that all inputs are within the spectrum of valid input data. The input vectors differ only in the secret input s. For instance, to assess the leakage of a secret RSA key, two valid secret RSA keys are generated randomly.

In the *sample collection* step, the implementation *imp* is run on the two input vectors that were generated in the previous step. For both runs, the observation made under the attacker model a is recorded. This step is repeated multiple times to obtain a collection of observations for each input vector.

In the *result computation* step, the arithmetic means of the two collections of observations are computed. For each collection, the frequency with which each observation occurs in the collection is computed and visualized in a histogram.

The last step is the *result evaluation*. Based on the computed results, one can detect weaknesses in implementations (if the means are clearly distinguishable and the histograms have little overlap). In addition to such qualitative results, quantitative results can be obtained through a statistical test (see Sect. 5).

3.2 Attacker Models

The sample-collection step in a distinguishing experiment depends on the attacker model. We implement this phase for two attacker models that we call *sequential* and *concurrent*. In both models, the attacker can execute an attack procedure with standard capabilities on the machine running the victim program. On Linux, attackers under both models can access powercap's pseudo-files on file system /sys. The model *sequential* captures active attackers who can trigger runs of the victim program. The model *concurrent* captures passive attackers who observe existing runs of the victim program. On Linux, unprivileged attackers can access information about running processes through file system /proc.

Implementation for Sequential. We implemented the measurement procedure for *sequential* in Python. Figure 3 shows corresponding pseudocode.

Firstly (Line 2), the attacker reads the energy-consumption counter through powercap by calling the function readCounter. Secondly, the attacker waits busily for the first change to the energy-consumption counter (Lines 3–5). Once the counter has been refreshed, the attacker invokes an execution of the victim program (Line 6) using the invocation command supplied as input to the attack procedure. After executing the victim program, the attacker queries the energy-consumption counter again (Line 7). The difference between the values of the

```
function readCounter
    $val \leftarrow$ read /sys/class/powercap/intel-rapl/intel-rapl:0/energy_uj
    return toInteger($val$)
end function
```

```
Parameters: $cmdLine$
$E_{instant} \leftarrow$ readCounter()
repeat                      ▷ Align beginning of measurement with register update
    $E_{begin} \leftarrow$ readCounter()
until $E_{begin} \neq E_{instant}$
invoke($cmdLine$)                                    ▷ Execute victim program
$E_{end} \leftarrow$ readCounter()
if $E_{end} < E_{begin}$ then
    discard measurement                              ▷ A wraparound has occurred
else
    return $E_{end} - E_{begin}$
end if
```

Fig. 3. Measurement procedure under *sequential*

```
Parameters: $victimComm$          ▷ the command name of the victim program
function waitForVictim
    while $true$ do
        $lastpid \leftarrow$ fifth field of /proc/loadavg
        repeat
            $newlastpid \leftarrow$ fifth field of /proc/loadavg
        until $lastpid \neq newlastpid$
        $pid \leftarrow lastpid$
        while $pid \leq newlastpid$ do
            $comm_{pid} \leftarrow$ contents of /proc/$pid$/comm
            if $comm_{pid} = victimComm$ then
                return $pid$
            end if
            $pid \leftarrow pid + 1$
        end while
    end while
end function
$pid \leftarrow$ waitForVictim()
$E_{begin} \leftarrow$ readCounter()
while /proc/$pid$/ exists do
    do nothing
end while
$E_{end} \leftarrow$ readCounter()
if $E_{end} < E_{begin}$ then
    discard measurement                              ▷ A wraparound has occurred
else
    return $E_{end} - E_{begin}$
end if
```

Fig. 4. Measurement procedure under *concurrent*

counter before and after the victim's execution is the attacker's sample. If the sample is negative, that is, if there was a wraparound of the counter, the sample is discarded (Lines 8–9). Otherwise, the sample is returned (Lines 10–11).

Implementation for Concurrent. Since an attacker under *concurrent* cannot trigger the victim program himself, he needs to identify runs of the victim program on the system. We use Python to implement the measurement procedure under *concurrent*. Pseudocode for the procedure is shown in Fig. 4.

The attacker waits until the victim program is executed (Lines 2–17). He detects the invocation of a program by monitoring the /proc filesystem. He recognizes the victim program by the command that was used to invoke it (Line 11). Once the victim program is executed, the attacker measures the energy consumption as the difference in the energy-consumption counter (Lines 19–27).

4 Qualitative Results on Bouncy Castle RSA

We investigate the consequences of software-based energy measurement on software security at the example of Intel RAPL and Bouncy Castle RSA. Using a distinguishing experiment, we identify that running Bouncy Castle RSA on a system with Intel RAPL gives rise to a weakness. The energy consumption of the decryption operation allows to distinguish between secret RSA keys. In the following, we describe the setup and results of our experiment in detail.

4.1 Experimental Setup

Assessed Implementation. To assess the vulnerability of Bouncy Castle RSA, we implement a Java program, called RSA, that decrypts an RSA ciphertext using Bouncy Castle 1.53. It takes a secret key and a ciphertext as input. It decrypts the ciphertext, using the secret key, and returns the resulting plaintext.

Figure 5 lists the pseudo-code of the program. Line 4 decrypts ciphertext *ct* using secret key *(d,n)*. processBlock is a method from Bouncy Castle's RSAEngine class, which implements the RSA decryption.

```
1: Input: $(d, n), ct$
2: $rsa \leftarrow$ New RSAEngine()
3: $rsa.\textsc{init}(false, (d, n))$
4: $result \leftarrow rsa.$
   $\textsc{processBlock}(ct, 0, ct.length)$
5: return $result$
```

Fig. 5. RSA decryption

Machine Configuration. We conduct our experiments on a Lenovo ThinkCentre M93p featuring one RAPL-capable Intel i5-4590 CPU @ 3.30GHz with 4GB of RAM. The machine runs Ubuntu 14.10 with a Linux kernel version 3.16.0-44-generic from Ubuntu's repository. The programs are executed using an OpenJDK 7 64-bit server Java Virtual Machine version 7u79-2.5.5-0ubuntu0.14.10.2 from Ubuntu's repository. To simulate a server machine that is shared between attacker and victim, we disable the X-server.

Parameters and Sampling. We generate two RSA keys $k1$ and $k2$ to supply as input to our RSA decryption program during our distinguishing experiment.

First, we randomly select two 1536 bit primes p and q to calculate the 3072 bit modulus $n = p * q$ shared by our keys. To select private exponents for the two keys $k1$ and $k2$, we exploit that $d * e \equiv 1 \pmod{(p-1)*(q-1)}$ must hold for valid RSA keys [45]. For $k1$, we randomly generate a public exponent e_{k1} and calculate the corresponding private exponent d_{k1}. For $k2$, we fix the public exponent to $e_{k2} = 65537$ and calculate the corresponding private exponent d_{k2}. The secret exponents that we obtain for $k1$ and $k2$ have Hamming weight 1460 and 1514, respectively. In addition to the keys, we randomly select a ciphertext $c < n$ to decrypt with both keys.

In our distinguishing experiments, we utilize our measurement procedures to collect 100000 samples per secret key under the attacker models *sequential* and *concurrent*. For the attacker model *concurrent*, under which an attacker cannot trigger executions of the victim program himself, we invoke the victim program after random delays between 100 ms and 1000 ms.

We reject outliers that lie further than six median absolute deviations from the median. For $k1$, we reject 1.24% of the samples under *sequential*, and 10.78% of the samples under *concurrent*. For $k2$, we reject 1.11% of the samples under *sequential*, and 11.01% of the samples under *concurrent*. We plot the collected samples for each key and attacker model as histograms.

4.2 Results for *Sequential*

The samples collected in our distinguishing experiment under *sequential* are depicted in Fig. 6. One histogram of energy-consumption samples is given per input. The histograms are colored based on the input: The blue (left) histogram corresponds to the samples for $k1$ with Hamming weight 1460, and the red (right) histogram corresponds to the samples for $k2$ with Hamming weight 1514.

The estimated mean energy consumption for $k1$ is $5.07J$, and for $k2$ the estimated mean energy consumption is $5.14J$. The peaks of the histograms and the mean energy consumptions for the inputs are clearly distinct.

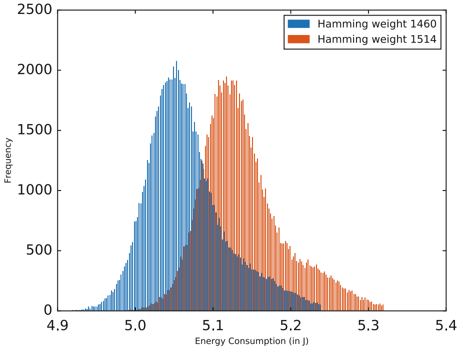

Fig. 6. Results for *sequential*

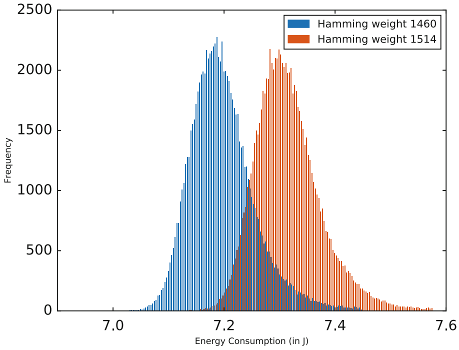

Fig. 7. Results for *concurrent*

Based on the histograms, an attacker under the model *sequential* can distinguish between the two secret RSA keys. Hence, there is a weakness in Bouncy Castle RSA in the presence of the Intel RAPL feature.

4.3 Results for *Concurrent*

Figure 7 shows the histograms of the samples per key under *concurrent*. Again, the blue (left) histogram corresponds to $k1$ (Hamming weight 1460) and the red (right) histogram corresponds to $k2$ (Hamming weight 1514).

The mean energy consumptions are $7.20J$ and $7.32J$ for the keys with Hamming weights 1460 and 1514, respectively. The peaks of the two histograms are clearly distinct. Interestingly, the overlap of the histograms is even a bit smaller compared to the overlap of the histograms under *sequential*. We will get back to this peculiarity in Sect. 5.

The mean energy consumptions and the histograms for the two RSA keys are clearly distinct. This means that the weakness we detected in Bouncy Castle RSA is even exposed to the weaker attacker model *concurrent*, under which an attacker only passively observes an RSA decryption.

Remark 1. Note that, the energy consumption measured under *concurrent* increased significantly by $2.13J$ and $2.18J$, respectively, compared the observations under *sequential*. This increase is due to the attacker actively monitoring the /proc filesystem to identify termination of the RSA process.

Overall, we identify a weakness in Bouncy Castle RSA that is exposed to attackers under, both, *sequential* and *concurrent*. For both attacker models, the mean energy consumption of the decryption differs significantly across the two RSA keys. Based on the histograms from our distinguishing experiments, an attacker is able to clearly distinguish between the two secret keys if he collects enough samples. In the following section, we quantify exactly how many samples an attacker needs in order to be successful.

5 Quantification of the Weakness

The results of our distinguishing experiments show that it is intuitively possible that an attacker can distinguish RSA keys by exploiting a weakness in Bouncy Castle RSA via Intel RAPL. We further investigate the likelihood of an attacker to distinguish keys. To this end, we devise a test procedure that allows an attacker to guess which of the two RSA key is used during decryption. Based on the false positive and false negative rates of the test procedure, we compute how many measurements an attacker requires to correctly guess the key in 99% of all cases.

5.1 A Distinguishing Test

Side-channel attacks, e.g., [8,13], can be mounted in two phases. In the first phase, the attacker collects a set of offline observations through the side channel as reference point, possibly on a different machine with the same software and hardware setup as the machine he shares with the victim. During the second phase, the attacker collects a set of online observations on the machine he shares with the victim. By relating his online side-channel observations with the offline observations, the attacker deduces information about the secret being processed.

For our distinguishing experiment setting, the offline observations are the collected energy-consumption characteristics of the RSA decryption operation for both, $k1$ and $k2$. The online observations would be side-channel observations collected to identify which key is used during a system run. To guess which key the system is using, the attacker compares how likely the learned energy-consumption characteristics allow him to explain the online observations. We model the guess by a statistical test to distinguish between the keys.

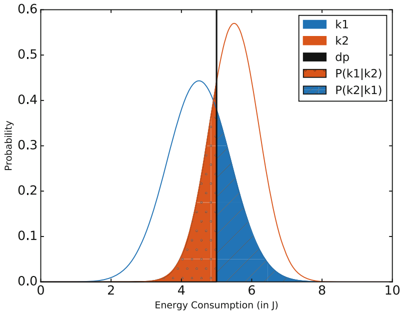

Fig. 8. Example of a distinguishing test

The attacker's distinguishing test works as follows: Given two keys, $k1$ and $k2$, with mean energy consumptions of m_{k1} and m_{k2}, where $m_{k1} < m_{k2}$, the attacker determines a *distinguishing point* $dp = \frac{m_{k1}+m_{k2}}{2}$. If the attacker observes an energy consumption less than dp, he guesses $k1$. Otherwise, he guesses $k2$. A false positive is: $k2$ was used but the attacker guesses $k1$. A false negative is: $k1$ was used but the attacker guesses $k2$.

A visualization of an example for the test is given in Fig. 8. In the example, the distributions of energy consumptions for $k1$ and $k2$ follow the normal distributions $\mathcal{N}(4.5J, 0.81)$ and $\mathcal{N}(5.5J, 0.49)$. Thus, the decision point is at $5J$. The area under the curve $k2$ to the left of dp corresponds to the false positive probability $P(k1|k2) = 23.75\%$. Conversely, the area under the curve $k1$ to the right of dp corresponds to the false negative probability $P(k2|k1) = 28.93\%$.

The attacker can use majority voting to increase his chances of guessing the correct key. For this, he observes multiple decryption operations and uses his test on each observation. Based on the individual guesses, he chooses the key on which the majority of guesses agreed. Let n be the number of observations the attacker makes. Then the false positive probability is $p^n_{P(k1|k2)} = P_{n,P(k1|k2)}(r > \lfloor \frac{n}{2} \rfloor)$ and the false negative probability is $p^n_{P(k2|k1)} = P_{n,P(k2|k1)}(r > \lfloor \frac{n}{2} \rfloor)$. Based on $P(k1|k2)$ and $P(k2|k1)$, one can determine the number n of observations needed for the attacker to distinguish $k1$ and $k2$ with 99% success rate, i.e., with $p^n_{P(k1|k2)} < 1\%$ and $p^n_{P(k2|k1)} < 1\%$. In the example from Fig. 8, $P(k1|k2) = 23.75\%$, so that 17 observations lead to a false positive rate $p^{17}_{P(k1|k2)} = 0.87\% < 1\%$. Conversely, $P(k2|k1) = 28.93\%$, so that 29 observations lead to a false negative rate below 1%, namely $p^{29}_{P(k2|k1)} = 0.81\%$. We conclude that the attacker requires 29 observations to distinguish $k1$ and $k2$ successfully in 99% of all cases.

5.2 Quantitative Results

For a quantitative evaluation of the weakness in Bouncy Castle RSA, we need to know the false positive and false negative probabilities of the distinguishing test. We estimate the probabilities based on the energy consumption characteristics collected offline by the attacker on his reference system. To estimate $P(k1|k2)$, we count the number of offline observations below dp of decryption samples with $k2$ and divide them by the total number of offline observations for $k2$. Conversely, to estimate the false negative probability we count the number of offline observations above dp of decryption samples of $k1$ and divide them by the total number of offline observations for $k1$. Formally, the probabilities can be estimated as follows. Let O_{k1} be the set of all offline observations for decryption operations with $k1$ and let O_{k2} be the set of all offline observations for $k2$.

$$P(k1|k2) = \frac{|\{x|x \in O_{k2} \wedge x < dp\}|}{|O_{k2}|} \qquad P(k2|k1) = \frac{|\{x|x \in O_{k1} \wedge x \geq dp\}|}{|O_{k1}|}$$

We evaluate the weakness for the attacker models *sequential* and *concurrent*, using our distinguishing test. For *sequential*, the distinguishing point is at $dp = 5.10J$, due to the means for $k1$ and $k2$ being $5.07J$ and $5.14J$, respectively (see Sect. 4.2). For *concurrent*, the distinguishing point is at $dp = 7.26J$, due to the means for $k1$ and $k2$ being $7.20J$ and $7.32J$, respectively (see Sect. 4.3).

The table in Fig. 9 lists the false positive and false negative probabilities $p^n_{P(k1|k2)}$ and $p^n_{P(k1|k2)}$ that result from n online observations for a given n under the two attacker models, respectively. Note that, the following equations hold: $P(k1|k2) = p^1_{P(k1|k2)}$ and $P(k2|k1) = p^1_{P(k2|k1)}$. In addition to $p^1_{P(k1|k2)}$ and $p^1_{P(k2|k1)}$, we only list the cases in which one of the probabilities falls below 1% for the first time. We highlight the first value below 1% for each of the probabilities by printing it in bold face.

The false positives for 1 observation range from 13.69% for *concurrent* to 24.75% for *sequential*. The false negatives for 1 observation range from 13.39% for *concurrent* to 19.77% for *sequential*. For 7 online observations, the false positive and false negative probabilities fall below 1% for *concurrent*. For *sequential*, the false negative probability falls below 1% at 13 observations and the false positive probability falls below 1% at 19 observations.

The distinguishing tests show that, in the worst case, only 19 observations are required to distinguish key $k1$ from key $k2$ in 99% of all cases. In this case of 19

	n	1 observation	7 observations	13 observations	19 observations
sequential	$p^n_{P(k1\|k2)}$	24.75%	6.83%	2.30%	**0.83**%
	$p^n_{P(k2\|k1)}$	19.77%	3.20%	**0.66**%	0.14%
concurrent	$p^n_{P(k1\|k2)}$	13.69%	**0.87**%	0.07%	0.007%
	$p^n_{P(k2\|k1)}$	13.39%	**0.80**%	0.06%	0.005%

Fig. 9. False-positive and false-negative rates for attackers

observations, *concurrent*'s test exhibits false negative and false positive probabilities below 0.01% each. This means that, given only 19 decryption observations, *concurrent* can distinguish both keys in 99.99% of all cases. Moreover, to distinguish both keys in 99% of all cases, *concurrent* requires only 7 observations. The finding that *concurrent*, our weakest attacker model, can distinguish both keys with high likelihood at 7 observations and, even worse, with near certainty at 19 observations, gives us reason to classify the weakness we discovered as severe.

Remark 2. A comparison across the two attacker models yields the surprising result that *concurrent* requires fewer observations than *sequential* to distinguish both keys in 99% of the cases. The 7 observations required by an attacker under *concurrent* are less than half of the 19 observations required by an attacker under *sequential*. Intuitively, an attacker under *sequential* should be able to distinguish the keys more easily than an attacker under *concurrent*, due to *sequential*'s ability to trigger victim executions and, hence, to measure more precisely.

After investigating the histograms from Sect. 4 again, our explanation is as follows. For both attacker models, *sequential* and *concurrent*, the overlap between both histograms seems to be roughly $0.25J$ wide. The estimated means differ by $0.07J$, and $0.12J$, respectively. While the width of the overlap remains similar with decreasing attacker capabilities, the means move further apart, decreasing the likelihood to observe an energy consumption value that lies in the overlap. Hence, the likelihood of an error in the distinguishing test decreases from *sequential* to *concurrent*, which is also shown by our quantitative results.

6 A Security Evaluation of Candidate Countermeasures

As we have shown in the previous sections, software-based energy side channels are a serious threat. Restricting access to software-based energy measurement features like Intel RAPL would seriously limit green IT. In contrast, software-level countermeasures would provide more flexibility, allowing energy measurement while mitigating information leakage through energy side channels.

We investigate two candidate software-level countermeasures, namely cross-copying [2] and conditional assignment [40]. Both are countermeasures against timing side channels, which ensure that equal or equivalent statements are executed across every pair of secret-dependent branches, independently of the guard. Intuitively, equal or equivalent statements should consume equivalent amounts of energy. Thus, we consider both techniques promising candidates for mitigating software-based energy side channels. In the following, we evaluate their effectiveness, using experiments and information theory.

6.1 Case Study

To investigate whether cross-copying or conditional assignment can help to mitigate leakage through software-based energy side channels, we quantify their effectiveness on a benchmark program. Motivated by the weakness that we detected

```
1: Input: $(d, n), c$
2: $r \leftarrow 1$
3: for $i = 1$ to $i = bitLength(d)$ do
4:     if $d \,\%\, 2 == 1$ then
5:         $r \leftarrow (r * y) \,\%\, n$
6:     else $r_{dummy} \leftarrow (r_{dummy} * y) \,\%\, n$
7:     end if
8:     $y \leftarrow (y * y) \,\%\, n$
9:     $d \leftarrow d >> 1$
10: end for
11: return $r \,\%\, n$
```

Fig. 10. Cross-copied version

```
1: Input: $(d, n), c$
2: $r \leftarrow 1$
3: for $i = 1$ to $i = bitLength(d)$ do
4:     $mask \leftarrow \sim(((d\%2 - 1) >> 31) \mid ((1 - d\%2) >> 31))$
5:     $r \leftarrow (mask \;\&\; ((r * y)\%n)) \mid (\sim mask \;\&\; r)$
6:     $y \leftarrow (y * y) \,\%\, n$
7:     $d \leftarrow d >> 1$
8: end for
9: return $r \,\%\, n$
```

Fig. 11. Conditional-assignment version

in the Bouncy Castle RSA implementation, we use a benchmark that is relevant for RSA. More concretely, we focus on an implementation of square-and-multiply modular exponentiation (Fig. 1).

We first check that software-based energy-side-channel leakage is a concern for this benchmark implementation. To this end, we approximate the channel capacity for the implementation. In the next step, we check whether the candidate countermeasures mitigate this threat. To this end, we approximate the channel capacity of a cross-copied version of the implementation and of a conditional-assignment version of the implementation. We evaluate the effectiveness of each countermeasure by the reduction in channel capacity that it causes.

The cross-copied implementation, shown in Fig. 10, contains a dummy assignment (Line 6) in the else-branch that is equivalent to the assignment in the then-branch. The conditional-assignment version replaces the branching by assignments masked by the branching condition (Fig. 11, Line 4 and 5).

6.2 Experimental Setup

For brevity, we call the unmitigated square-and-multiply implementation *Baseline*, the cross-copied implementation *CC*, and the conditional-assignment version *CA*. For experimental evaluation, we use [36]'s Java implementation of Baseline, CC, and CA. We adapt the implementations to log the energy consumption measured through powercap. We disable the network and all but the first CPU core to reduce noise in the measurements. We disable the just-in-time (JIT) compiler of the Java VM to prevent optimizations from interfering with our results. To avoid zero energy consumption results due to execution times below 1 ms, we repeat the computation 1.31×10^5 times. This results in approximately 100 updates of the energy-consumption counter for a single execution of Baseline. We estimate the channel capacity using an iterative Blahut-Arimoto algorithm [5,10] based on the samples collected during a distinguishing experiment.

For the distinguishing experiment, we use two input vectors that share $n = 4096$ and $c = 1234567890$. One secret exponent with Hamming weight 5 ($d = 2080374784$) and one secret exponent with Hamming weight 25 ($d = 33554431$)

are used as the first and second value of the secret input, respectively. We follow [36] and collect 10000 samples per input. We reject outliers that lie further than six median absolute deviations from the median. This results in a rejection between 1.07% and 2.73% of all samples for each implementation and each input.

6.3 Experimental Results and Interpretation

The table in Fig. 12 shows the results of our experiments. The mean energy consumptions and channel capacities are given with 95% confidence intervals.

The mean energy consumption for the first input to Baseline is roughly $15373.73nJ$. The mean energy consumption for the second input to Baseline is roughly $18934.13nJ$. These means are clearly distinguishable. Hence, there is a clear security concern already in the benchmark.

To quantify the severity of the security threat, we determine the channel capacity. Since we consider a scenario in which the attacker tries to distinguish between two inputs, the secret is 1 bit, namely the choice of the input. For Baseline, $C(X;Y)$ is 0.9922 bits/symbol. That is, one attacker observation reveals almost the entire secret under the worst-case prior distribution of inputs.

Next, we investigate the results for CC. Here, the mean energy consumptions for the two inputs are roughly $20372.21nJ$ and $21040.05nJ$, respectively. The channel capacity is approximately 0.9171 bits/symbol.

Intuitively, the mean energy consumptions of CC are still clearly distinguishable. The quantification of the security concern by the channel capacity of CC confirms that the concern is still substantial. CC can still leak 91% of the secret under the worst-case prior input distribution. This shows that [36]'s cross-copying implementation does not mitigate the energy side channel significantly.

We can only speculate why cross-copying is not effective against the energy side channel in our experiments. The difference of data dependencies introduced by the branches might be responsible. In the *else* branch (Fig. 10, Line 6), the result is written to r_{dummy} instead of r. This might cause a subtle difference in energy consumption, for example, due to different patterns of pipeline stalling.

Next, we investigate the results for CA. The mean energy consumptions of CA for the two inputs are roughly $32670.41nJ$ and $32630.73nJ$, respectively. The channel capacity is approximately 0.0075 bits/symbol.

The mean energy consumptions for the two inputs to CA are almost identical and, hence, not easy to distinguish. The channel capacity is reduced almost to zero. That is, in our example, conditional assignment effectively reduces the security concern by 99%, almost eliminating the software-based energy side channel.

The successful reduction of channel capacity from Baseline to CA gives us hope that an effective countermeasure against software-based energy side channels can be designed. In particular, conditional assignment is a promising starting point in the design of such countermeasures.

	$mean(E)$ for Input 1	$mean(E)$ for Input 2	$C(X;Y)$
Baseline	$15370.07nJ \pm 3.18$	$18925.46nJ \pm 4.00$	0.9922 ± 0.00000
CC	$20372.21nJ \pm 4.48$	$21040.05nJ \pm 3.97$	0.9171 ± 0.00970
CA	$32670.41nJ \pm 5.63$	$32630.73nJ \pm 5.60$	0.0075 ± 0.00375

Fig. 12. Statistical results for modular exponentiation

7 Related Work

7.1 Power-Consumption Side Channels

Power-consumption side channels are exploited, e.g., by the techniques *Simple Power Analysis* (SPA) and *Differential Power Analysis* (DPA). These techniques were introduced by Kocher, Jaffe, and Jun in attacks on smartcards implementing the DES cryptosystem [23]. In both techniques, traces of the power consumption of a circuit are measured and analyzed. SPA is a direct interpretation of power traces and can yield information about a device's secret key during crypto computations [23,24]. DPA is a statistical method to identify correlations between data processed and power consumption [23,24]. Variations of power analysis have been used in attacks on implementations of cryptography, e.g., of DES [23,28,47], of RSA [23,24,37,42], and of AES [24,34,44]. All these attacks obtain traces of power consumption from measurements with dedicated hardware.

Recently, power-consumption side channels were exploited without dedicated hardware on mobile devices using batteries [38,51]. We briefly give an overview on Michalevsky et al.'s work on tracking Android devices through power analysis [38]. They measure the power consumption of a device using its battery monitoring unit. By their measurements, they can, e.g., track users in real-time.

Our work on software-based energy side channels differs from the previously described work on power analysis in the two following aspects.

(a) We investigate a fundamentally weaker attacker model. Our attacker is only able to measure the energy consumption, which is the aggregate of instantaneous power consumption. As a result, the observations required for an attack through software-based energy side channels are more coarse-grained.

(b) On the technical side, we use software-based measurement techniques available on machines without battery, e.g., on desktop and server machines. Software-based techniques allow an attacker to conduct his attack without dedicated hardware and without physical access to the device under attack. Thus, the observations required to exploit software-based energy side channels are easier to obtain than power traces and might be obtainable remotely in the cloud.

Overall, we think that software-based energy side channels are an interesting target for future security research because they use more coarse-grained observations that are easier to obtain.

7.2 Quantitative Side-Channel Analysis

Side channels have been the focus of many research projects since their first appearance in Kocher's work in 1996 [22]. A multitude of work focuses on exploiting side channels, e.g., [3,4,8,11,22,32,52]. In addition, analysis of side channels using information-theoretic methods has become an area of focus. Köpf and Basin propose a model to analyze adaptive side-channel attacks using information theory [25]. More concretely, they quantify the attacker's uncertainty about a secret based on the number of side-channel measurements the attacker obtained. CacheAudit [18] by Doychev, Köpf, Mauborgne, and Reineke is a tool employing program analysis and information theory to give upper bounds on information leakage through cache side channels in x86 binaries. Other work on analysis of side channels using information theory includes [9,27,33,35,49].

The mentioned works are foremost of analytic nature. On the *empirical* analysis of side channels, we are aware of only few works, e.g., [14,16,36]. Mantel and Starostin evaluate the practical effectiveness of program transformations to mitigate timing side channels [36]. For their evaluation, they consider the capacity of the timing side channel in a program. They introduce the idea of distinguishing experiments to obtain experimental results on the side-channel capacity.

We apply [36]'s concept of distinguishing experiments to show software-based energy side channels exist. Following [36]'s approach, we use channel capacity to evaluate the effectiveness of side-channel countermeasures. In summary, we build on [36]'s techniques, but apply them to a novel type of side channel.

Our distinguishing test to quantitatively evaluate the weakness in Bouncy Castle RSA is a variant of [13]'s test to distinguish e-passports. Distinguishing e-passports is done through sending a random message and a replayed message to a passport to obtain the difference in response times. Using a normal distribution as a model of response times and a manually selected distinguishing point, Chothia and Smirnov calculate the number of observations needed to distinguish passports in 98% of all cases. We transfer the test to our setting. Unlike Chothia and Smirnov, we estimate error probabilities based on offline samples alone, because our observations do not follow a normal distribution.

Like the distinguishing attack in [3] and the distinguishing experiments in [16,36] we focus on distinguishing between two secrets in our qualitative and quantitative evaluation. We take care to use two representative secrets by following standard random key generation procedures (OpenSSL's default public exponent, criteria in [45]). A notable work that distinguishes between more than two secrets is [13], which considers ten different e-passports.

8 Conclusion

Software-based energy measurement features facilitate the optimization of energy consumption, which is crucial in datacenters. We showed, at the example of Intel RAPL and Bouncy Castle RSA, that these important features also introduce a security issue. Based on only 7 energy samples measured with Intel RAPL, an attacker can distinguish between two RSA secret keys with 99% success probability. Overall, our results show that software-based energy side channels are a serious security concern.

To protect against the security issues without excluding a large fraction of programs from the optimization of energy consumption, fine-grained countermeasures are needed. We have identified conditional assignment as a promising starting point for designing such countermeasures. In our quantitative experimental evaluation of candidate countermeasures, conditional assignment was effective in the protection of our benchmark program.

Interesting directions for future work will be to derive key-recovery attacks against Bouncy Castle RSA from our results and to investigate the effect of just-in-time compilation. We hope that our approach using distinguishing experiments will also be helpful for the timely detection of side-channel vulnerabilities in other security-critical implementations.

Acknowledgements. We thank the anonymous reviewers for their helpful comments. We thank Yuri Gil Dantas, Ximeng Li, and Artem Starostin for helpful suggestions at different stages of our research project. This work has been funded by the DFG as part of the project Secure Refinement of Cryptographic Algorithms (E3) within the CRC 1119 CROSSING.

A RSA Parameters

We list the ciphertext c, the modulus n, and, for each of $k1$ and $k2$, the private exponent d. The table in Fig. 13 lists the bit length and Hamming weight of the individual key parameters.

Variable	Bit Length	Hamming Weight
n	3071	1550
d_{k1}	2880	1460
d_{k2}	3070	1514

Fig. 13. RSA parameter information

$$c = 21\,444\,858\,737\,899\,529\,054\,620\,511\,370\,454\,507\,092\,966\,801\,560\,642\,267\,256$$
$$271\,104\,479\,565\,623\,317\,752$$

$$\begin{aligned} n = {}& 2\,701\,439\,070\,847\,831\,436\,302\,643\,023\,883\,472\,860\,688\,598\,232\,186\,843\,078\,227 \\ & 336\,630\,239\,028\,012\,256\,550\,437\,650\,268\,769\,791\,198\,665\,992\,795\,439\,484\,217 \\ & 556\,231\,560\,025\,070\,371\,698\,339\,396\,459\,200\,881\,954\,828\,050\,340\,830\,157\,513 \\ & 508\,421\,214\,770\,279\,402\,829\,167\,697\,307\,613\,566\,394\,176\,659\,624\,110\,756\,710 \\ & 628\,073\,014\,761\,357\,607\,996\,466\,364\,229\,898\,558\,058\,073\,647\,928\,107\,882\,490 \\ & 406\,530\,947\,890\,797\,815\,573\,279\,825\,845\,151\,878\,854\,668\,533\,049\,684\,979\,849 \\ & 046\,263\,217\,739\,454\,991\,182\,947\,451\,853\,315\,650\,216\,590\,304\,861\,483\,322\,060 \\ & 060\,830\,631\,094\,083\,537\,687\,041\,942\,037\,690\,007\,693\,207\,305\,415\,195\,214\,688 \\ & 380\,836\,084\,216\,172\,144\,792\,635\,213\,107\,935\,419\,683\,137\,307\,723\,939\,160\,685 \\ & 162\,963\,798\,575\,432\,937\,877\,504\,919\,069\,927\,206\,463\,822\,812\,215\,130\,775\,583 \\ & 846\,864\,507\,114\,293\,297\,396\,044\,572\,999\,463\,005\,723\,946\,293\,357\,342\,314\,317 \\ & 073\,651\,823\,518\,140\,604\,749\,430\,721\,177\,242\,193\,915\,300\,702\,995\,100\,318\,209 \\ & 072\,680\,035\,930\,026\,760\,088\,409\,999\,868\,552\,738\,596\,292\,995\,373\,879\,363\,788 \\ & 033\,672\,926\,557\,820\,859\,907\,396\,638\,610\,163\,158\,192\,481\,639\,061\,519\,053\,725 \\ & 943\,865\,537\,221\,937\,014\,172\,943\,369\,946\,317\,527\,944\,500\,414\,286\,628\,781\,268 \\ & 545\,323\,413\,089\,483\,205\,130\,985\,579\,709\,706\,141\,004\,772\,358\,028\,235\,835\,383 \\ & 909\,088\,091\,781 \end{aligned}$$

$$\begin{aligned} d_{k1} = {}& 834\,165\,241\,298\,999\,430\,572\,239\,556\,741\,255\,001\,409\,654\,369\,991\,231\,022\,229 \\ & 220\,766\,012\,080\,697\,463\,656\,309\,174\,093\,432\,158\,675\,603\,340\,216\,003\,665\,704 \\ & 131\,245\,121\,040\,967\,995\,188\,366\,594\,646\,886\,723\,499\,562\,164\,775\,785\,136\,008 \\ & 896\,297\,468\,405\,676\,356\,520\,936\,826\,945\,820\,428\,827\,348\,255\,217\,929\,032\,541 \\ & 402\,713\,897\,358\,199\,944\,878\,768\,362\,082\,394\,995\,264\,828\,906\,821\,922\,160\,081 \\ & 896\,178\,733\,905\,626\,880\,183\,545\,477\,730\,549\,240\,816\,967\,899\,639\,830\,638\,962 \\ & 585\,672\,589\,316\,902\,773\,646\,421\,798\,550\,172\,445\,107\,122\,780\,716\,202\,671\,225 \\ & 380\,537\,248\,843\,847\,787\,001\,886\,230\,297\,573\,272\,017\,826\,827\,441\,391\,799\,971 \\ & 383\,481\,609\,479\,693\,434\,609\,255\,364\,781\,237\,298\,674\,935\,211\,620\,000\,100\,041 \\ & 121\,931\,493\,922\,732\,461\,726\,369\,423\,008\,396\,966\,929\,501\,865\,211\,495\,345\,778 \\ & 306\,377\,790\,415\,705\,746\,828\,081\,157\,687\,854\,396\,051\,014\,887\,511\,709\,430\,472 \\ & 332\,036\,102\,915\,852\,198\,291\,900\,816\,398\,410\,487\,823\,293\,583\,922\,839\,328\,518 \\ & 348\,451\,707\,669\,403\,333\,993\,535\,972\,295\,702\,111\,655\,470\,282\,959\,323\,284\,437 \\ & 483\,178\,409\,938\,904\,891\,941\,353\,380\,152\,662\,307\,486\,605\,772\,459\,905\,400\,151 \\ & 595\,208\,101\,373\,686\,515\,401\,901\,692\,964\,058\,539\,933\,630\,431\,256\,790\,357\,003 \\ & 951\,566\,054\,871 \end{aligned}$$

$$\begin{aligned} d_{k2} = {} & 849\,669\,096\,348\,419\,204\,365\,570\,298\,477\,349\,071\,171\,614\,131\,865\,471\,357\,729 \\ & 223\,033\,692\,678\,706\,938\,741\,080\,172\,802\,999\,095\,258\,832\,447\,464\,674\,826\,253 \\ & 513\,078\,126\,047\,832\,149\,347\,969\,391\,019\,019\,909\,054\,959\,345\,128\,332\,576\,053 \\ & 617\,789\,744\,725\,266\,175\,298\,192\,375\,980\,008\,826\,221\,571\,989\,636\,873\,751\,134 \\ & 110\,143\,415\,982\,969\,381\,778\,707\,618\,076\,367\,532\,496\,926\,501\,132\,827\,071\,452 \\ & 381\,857\,918\,868\,318\,894\,249\,233\,517\,709\,784\,025\,494\,473\,083\,475\,794\,688\,338 \\ & 318\,669\,205\,292\,634\,477\,215\,223\,397\,852\,394\,761\,705\,823\,824\,009\,487\,094\,582 \\ & 053\,403\,448\,414\,519\,187\,059\,874\,506\,785\,829\,441\,820\,347\,012\,931\,983\,749\,032 \\ & 937\,029\,535\,204\,674\,669\,118\,349\,387\,871\,614\,945\,298\,028\,125\,580\,430\,251\,234 \\ & 668\,630\,080\,219\,358\,718\,245\,352\,291\,415\,465\,763\,013\,100\,923\,209\,592\,436\,665 \\ & 013\,250\,115\,828\,673\,733\,662\,998\,810\,262\,212\,481\,440\,283\,643\,807\,643\,936\,814 \\ & 117\,781\,430\,012\,258\,146\,460\,658\,672\,860\,115\,805\,136\,484\,154\,272\,106\,257\,859 \\ & 724\,501\,287\,380\,315\,081\,559\,737\,344\,179\,353\,409\,746\,394\,603\,117\,859\,928\,408 \\ & 887\,186\,955\,223\,875\,953\,551\,569\,984\,766\,380\,086\,437\,972\,232\,285\,448\,676\,372 \\ & 452\,773\,194\,118\,503\,147\,494\,678\,742\,399\,709\,855\,779\,414\,952\,984\,145\,813\,209 \\ & 160\,450\,714\,556\,753\,389\,051\,248\,506\,613\,925\,218\,229\,813\,615\,602\,923\,271\,485 \\ & 462\,745\,822\,621 \end{aligned}$$

References

1. Acıiçmez, O., Koç, Ç.K., Seifert, J.-P.: Predicting secret keys via branch prediction. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 225–242. Springer, Heidelberg (2006). https://doi.org/10.1007/11967668_15
2. Agat, J.: Transforming out timing leaks. In: POPL, pp. 40–53 (2000)
3. AlFardan, N.J., Paterson, K.G.: Lucky thirteen: breaking the TLS and DTLS record protocols. In: S&P, pp. 526–540 (2013)
4. Andrysco, M., Kohlbrenner, D., Mowery, K., Jhala, R., Lerner, S., Shacham, H.: On subnormal floating point and abnormal timing. In: S&P, pp. 623–639 (2015)
5. Arimoto, S.: An algorithm for computing the capacity of arbitrary discrete memoryless channels. IEEE Trans. Inf. Theory **18**(1), 14–20 (1972)
6. Barthe, G., Rezk, T., Warnier, M.: Preventing timing leaks through transactional branching instructions. Electr. Notes Theor. Comput. Sci. **153**(2), 33–55 (2006)
7. Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 92–111. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0053428
8. Bernstein, D.J.: Cache-Timing Attacks on AES (2005)
9. Bindel, N., Buchmann, J., Krämer, J., Mantel, H., Schickel, J., Weber, A.: Bounding the cache-side-channel leakage of lattice-based signature schemes using program semantics. In: Imine, A., Fernandez, J.M., Marion, J.-Y., Logrippo, L., Garcia-Alfaro, J. (eds.) FPS 2017. LNCS, vol. 10723, pp. 225–241. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75650-9_15
10. Blahut, R.E.: Computation of channel capacity and rate-distortion functions. IEEE Trans. Inf. Theory **18**(4), 460–473 (1972)

11. Brumley, B.B., Tuveri, N.: Remote timing attacks are still practical. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 355–371. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23822-2_20
12. Chatzikokolakis, K., Chothia, T., Guha, A.: Statistical measurement of information leakage. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 390–404. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12002-2_33
13. Chothia, T., Smirnov, V.: A traceability attack against e-Passports. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 20–34. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14577-3_5
14. Cock, D., Ge, Q., Murray, T.C., Heiser, G.: The last mile: an empirical study of timing channels on seL4. In: CCS, pp. 570–581 (2014)
15. Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2nd edn. Wiley, Hoboken (2006)
16. Dantas, Y.G., Gay, R., Hamann, T., Mantel, H., Schickel, J.: An evaluation of bucketing in systems with non-deterministic timing behavior. In: IFIP SEC (2018, to appear)
17. David, H., Gorbatov, E., Hanebutte, U.R., Khanna, R., Le, C.: RAPL: memory power estimation and capping. In: ISLPED, pp. 189–194 (2010)
18. Doychev, G., Köpf, B., Mauborgne, L., Reineke, J.: CacheAudit: a tool for the static analysis of cache side channels. ACM Trans. Inf. Syst. Secur. **18**(1), 4:1–4:32 (2015)
19. Farkas, K.I., Flinn, J., Back, G., Grunwald, D., Anderson, J.M.: Quantifying the energy consumption of a pocket computer and a Java virtual machine. In: SIGMETRICS, pp. 252–263 (2000)
20. Hähnel, M., Döbel, B., Völp, M., Härtig, H.: Measuring energy consumption for short code paths using RAPL. SIGMETRICS Perform. Eval. Rev. **40**(3), 13–17 (2012)
21. Intel: Intel-64 and IA-32 Architectures Software Developer's Manual. Volume 3 (3A, 3B, & 3C): System Programming Guide (2017)
22. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
23. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
24. Kocher, P.C., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. J. Cryptogr. Eng. **1**(1), 5–27 (2011)
25. Köpf, B., Basin, D.A.: An information-theoretic model for adaptive side-channel attacks. In: CCS, pp. 286–296 (2007)
26. Köpf, B., Mantel, H.: Transformational typing and unification for automatically correcting insecure programs. Int. J. Inf. Sec. **6**(2–3), 107–131 (2007)
27. Köpf, B., Smith, G.: Vulnerability bounds and leakage resilience of blinded cryptography under timing attacks. In: CSF, pp. 44–56 (2010)
28. Ledig, H., Muller, F., Valette, F.: Enhancing collision attacks. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 176–190. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_13
29. Legion of the Bouncy Castle Inc.: The Legion of the Bouncy Castle. https://www.bouncycastle.org/. Accessed 12 Apr 2018
30. Linux Kernel Organization Inc: Power Capping Framework. https://www.kernel.org/doc/Documentation/power/powercap/powercap.txt. Accessed 18 Apr 2018

31. Linux Programmer's Manual: MSR - x86 CPU MSR access device (2009). http://man7.org/linux/man-pages/man4/msr.4.html. Accessed 12 Apr 2018
32. Lipp, M., Gruss, D., Spreitzer, R., Maurice, C., Mangard, S.: Armageddon: cache attacks on mobile devices. In: USENIX Security, pp. 549–564 (2016)
33. Macé, F., Standaert, F.-X., Quisquater, J.-J.: Information theoretic evaluation of side-channel resistant logic styles. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 427–442. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_29
34. Mangard, S.: A simple power-analysis (SPA) attack on implementations of the AES key expansion. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 343–358. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36552-4_24
35. Mantel, H., Weber, A., Köpf, B.: A systematic study of cache side channels across AES implementations. In: Bodden, E., Payer, M., Athanasopoulos, E. (eds.) ESSoS 2017. LNCS, vol. 10379, pp. 213–230. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62105-0_14
36. Mantel, H., Starostin, A.: Transforming out timing leaks, more or less. In: Pernul, G., Ryan, P.Y.A., Weippl, E. (eds.) ESORICS 2015. LNCS, vol. 9326, pp. 447–467. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24174-6_23
37. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Power analysis attacks of modular exponentiation in smartcards. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 144–157. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_14
38. Michalevsky, Y., Schulman, A., Veerapandian, G.A., Boneh, D., Nakibly, G.: Powerspy: location tracking using mobile device power analysis. In: USENIX Security, pp. 785–800 (2015)
39. Millen, J.K.: Covert channel capacity. In: S&P, pp. 60–66 (1987)
40. Molnar, D., Piotrowski, M., Schultz, D., Wagner, D.: The program counter security model: automatic detection and removal of control-flow side channel attacks. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 156–168. Springer, Heidelberg (2006). https://doi.org/10.1007/11734727_14
41. Noureddine, A., Rouvoy, R., Seinturier, L.: Monitoring energy hotspots in software - energy profiling of software code. Autom. Softw. Eng. **22**(3), 291–332 (2015)
42. Novak, R.: SPA-based adaptive chosen-ciphertext attack on RSA implementation. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 252–262. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45664-3_18
43. Page, D.: Theoretical use of cache memory as a cryptanalytic side-channel. IACR Cryptology ePrint Archive, pp. 1–23 (2002)
44. Renauld, M., Standaert, F.-X., Veyrat-Charvillon, N.: Algebraic side-channel attacks on the AES: why time also matters in DPA. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 97–111. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04138-9_8
45. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
46. RSA Laboratories: PKCS #1 v2.2: RSA Cryptography Standard (2012). https://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-standard-wp.pdf. Accessed 12 Apr 2018
47. Schramm, K., Wollinger, T., Paar, C.: A new class of collision attacks and its application to DES. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 206–222. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39887-5_16
48. Snedecor, G.W., Cochran, W.G.: Statistical Methods, 8th edn. Iowa State University Press, Ames (1989)

49. Standaert, F.-X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_26
50. Tyley, R.: Spongy Castle by rtyley. https://rtyley.github.io/spongycastle/. Accessed 12 Apr 2018
51. Yan, L., Guo, Y., Chen, X., Mei, H.: A study on power side channels on mobile devices. In: Internetware, pp. 30–38 (2015)
52. Yarom, Y., Falkner, K.: FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack. In: USENIX Security, pp. 719–732 (2014)