2015 IEEE 28th Computer Security Foundations Symposium

Rational Protection Against Timing Attacks

Goran Doychev
IMDEA Software Institute
goran.doychev@imdea.org

Boris Köpf
IMDEA Software Institute
boris.koepf@imdea.org

***Abstract*—Timing attacks can effectively recover keys from cryptosystems. While they can be defeated using constant-time implementations, this defensive approach comes at the price of a performance penalty. One is hence faced with the problem of striking a balance between performance and security against timing attacks.**

In this paper, we propose a systematic approach for determining the optimal protection against timing attacks, on the example of cryptosystems based on discrete logarithms. Our model includes a resource-bounded timing adversary who strives to maximize the probability of key recovery, and a defender who strives to reduce the cost while maintaining a certain degree of security. We obtain the optimal protection as an equilibrium in a game between the defender and the adversary. At the heart of the equilibrium computation are novel bounds for the probability of key recovery, which are expressed as a function of the applied protection and the attack strategy of a timing adversary.

We put our techniques to work in a case study in which we identify optimal protections for libgcrypt's ElGamal implementation. We determine situations in which the optimal choice is to use a defensive, constant-time implementation and a small key, and situations in which the optimal choice is a more aggressively tuned (but leaky) implementation with a longer key.

I. Introduction

Side-channel attacks break the security of systems by exploiting signals that are unwittingly emitted by the implementation. Examples of such signals are the consumption of power [21], memory [18], and execution time [20]. Execution time is a particularly daunting signal because it can be measured and exploited from a long distance [11], which opens the door for a potentially large number of attackers.

In theory, one can get rid of timing side channels by avoiding the use of secret-dependent control flow and of performance-enhancing features of the hardware architecture, such as caches and branch prediction units. However, this defensive approach comes at the price of a performance penalty. In practice, one is hence faced with the problem of striking a balance between performance and security against timing attacks.

In this paper we present a game-theoretic approach for solving this problem. The key novelty of our approach is a simple and practical model of the side-channel adversary as a player that can distribute the available resources between timing measurements and offline search for the secret. Our approach is focused in that we consider only cryptosystems based on discrete logarithms and input blinding as a countermeasure, yet it is comprehensive in that it goes all the way from formal modeling to identifying the optimal protection for a library implementation of ElGamal. A highlight of our results is that we are the first to formally justify the use of a fast but (slightly) leaky implementation over a defensive constant-time implementation, for some parameter ranges.

On a technical level, we identify the optimal countermeasure configuration as an equilibrium of a two-stage (Stackelberg) game between two rational players: an adversary and a defender. The adversary strives to maximize the probability of key recovery, by distributing bounded resources between timing measurements and computational search for the key. The defender strives to minimize the cost of protection, while maintaining a certain security level given in terms of an upper bound on the probability of adversary success. The defender's means to achieve this are the choice of the key length and the configuration of the countermeasure.

At the heart of the equilibrium computation are novel bounds for the probability of key recovery in the presence of side-channel information. We derive these bounds in the generic group model and under the assumption that the cryptosystem is based on discrete logarithms and protected against timing attacks by an idealized form of input blinding.

Our starting point is an existing upper bound for the amount of information contained in n timing measurements, when the execution time is discretized [23]. The technical challenge we face is to turn this bound into a guarantee against an adversary that can mount a combined timing/algebraic attack. We identify unpredictability entropy [17] as a suitable tool for this task. In particular, unpredictability entropy satisfies a chain rule [25], which limits the extent to which bounded leakage can decrease the hardness of a computational problem. We then cast Shoup's lower bound for computing discrete logs in generic groups [32] in terms of the unpredictability entropy w.r.t. an adversary who can perform m group operations. Finally, we connect the leakage bound with the bound for the discrete log to obtain the desired bound (in terms of n and m) for combined side channel/algebraic adversaries.

DOI 10.1109/CSF.2015.39

We put our approach to work in a case study where we seek to optimally configure countermeasures against timing attacks in libgcrypt's ElGamal implementation. Experimentally, we identify optimal choices of key lengths and countermeasure configurations that guarantee the same degree of security as a constant-time implementation using a reference key length. We do this for realistic server configurations, and target commonly used key lengths. In the course of our experiments we observe that the time of deployment of a key, or the number allowed connections per second can influence which configuration is optimal: the defensive, constant-time implementation with a short key, or the more aggressively tuned and leaky implementation with a longer key.

In summary, our contributions are conceptual and practical. Conceptually, we combine game theory with novel, quantitative security guarantees to tackle the problem of systematically choosing the optimal balance between security and performance. Practically, we perform a case-study on a realistic ElGamal implementation, where we illustrate how our techniques can be used to identify cost-effective countermeasure configurations.

Organization: In Section II, we present the countermeasure configuration game, based on parametric security guarantees. We instantiate these guarantees in Section III, before we discuss their application in Sections IV and V. We present related work in Section VI, and conclude in Section VII.

II. Choice of Optimal Protection

In this section we cast the optimal configuration of a countermeasure as a game between the adversary and the defender. We conclude the section with a discussion of the effect of using safe approximations of the security of a system (instead of exact values) for the solution of the game.

A. Motivating Example

Input blinding is a widely deployed countermeasure against timing attacks on cryptosystems based on modular exponentiation. A formal security analysis of input blinding [23] shows that the amount of information about the key that is leaked by a blinded cryptosystem is bounded from above by

$$(b-1)\log_2(n+1) \tag{1}$$

bits, where b is the number of possible execution times, and n is the number of side-channel measurements made.

For real systems, b can be as large as the difference between the worst-case execution time and the best-case execution time (e.g., in clock ticks), in which case (1) does not imply meaningful guarantees. However, b can be reduced by applying *bucketing*, which is the discretization of system's execution times into intervals (*buckets*) where, for each execution time, one waits until the enclosing bucket's upper bound before returning the result of the computation.

Choosing a smaller number of buckets b leads to better security guarantees, but it also leads to a decrease in performance. Likewise, picking a larger key size leads to better security and a decrease in performance. The techniques presented in this paper enable identifying the sweet spot in the resulting parameter space.

B. Countermeasure Configuration as a Game

We formalize the configuration of a countermeasure as a two-stage game between a defender ($\mathcal{D}$) and an adversary ($\mathcal{A}$). Similar games are known in the literature as *Stackelberg* games [14]. In the first stage, $\mathcal{D}$ chooses an element d from a finite set D of *defender actions*, which can be parameters of a protection mechanism. In the second stage, having seen d, $\mathcal{A}$ responds by choosing an element of a finite set A of *adversary actions*. We model this choice as a function $r\colon D \to A$, which captures that the adversary knows and responds to d.

If a situation is preferred by a player, we say that it has a higher *utility*. Utilities are captured in terms of real-valued functions of both player's actions:

$$u_{\mathcal{D}}, u_{\mathcal{A}}\colon D \times A \to \mathbb{R} \ .$$

Putting together all ingredients, a two-stage game is the tuple $G = [(\mathcal{D}, \mathcal{A}), (D, A), (u_{\mathcal{D}}, u_{\mathcal{A}})]$.

C. Utilities of the Players

Now we define the utilities that make up the countermeasure configuration game. For this, we rely on (upper bounds on) the probability of a breach (i.e. a successful attack), which we model as functions

$$p\colon D \times A \to [0,1] \ .$$

Throughout this section we leave p parametric; we show how to instantiate it in Section III.

Defender's utility: A natural notion for the defender's utility is to consider the loss l in case of a successful breach together with the cost of the defense d, i.e.,

$$-p(d, r(d)) \cdot l - cost_{\mathcal{D}}(d) \ . \tag{2}$$

In practice, this definition suffers from two problems: First, in advance it may be difficult to put a number l on the loss in case of breach. Second, the definition may give misleading results when working with *bounds* on p, as we discuss in Section II-E.

To address those problems, we impose a hard upper bound $p_{\max}$ on the probability of breach, which is common practice, for example, in safety requirements [1]. We then define the defender's utility as the cost of defense under this constraint.

Definition 1: The *defender's utility* $u_{\mathcal{D}}$ is defined as

$$u_{\mathcal{D}}(d, r(d)) = -cost_{\mathcal{D}}(d) \ ,$$

where we require $p(d, r(d)) \leq p_{\max}$.

Example 1: For the countermeasure described in Section II-A, the defender's action is to choose a key length k and a number of buckets b, such that the adversary's chance of key recovery remains below $p_{\max}$:

$$D = \{(k, b) \mid p((k, b), r(k, b)) \leq p_{\max}\} \ ,$$

where $cost_{\mathcal{D}}(k, b)$ is the average execution time of the corresponding implementation. Here we instantiate $p(\cdot)$ using the bounds in Section III. We rely on key length recommendations [2] for protecting against traditional (non-side channel) adversaries as a basis for instantiating $p_{\max}$. Specifically, the defender has to pick the key length k and number of buckets b in such a way that the security level *with* side channel matches that of a reference key length k_{ref} *without* side channel.

Adversary's utility: A natural notion for the adversary's utility is the adversary's expected benefits in terms of the gain g in case of a successful breach and the cost of attack, i.e.,

$$p(d, r(d)) \cdot g - cost_{\mathcal{A}}(r(d)) \ .$$

In practice, however, it is more common to capture adversaries in terms of assumptions on their resources and capabilities [2]. We follow this approach and impose a hard upper bound Δ on the attacker's resources. We then define the adversary's utility as the probability of breach under this constraint.

Definition 2: The *adversary's utility* $u_{\mathcal{A}}$ is defined as

$$u_{\mathcal{A}}(d, r(d)) = p(d, r(d)) \ ,$$

where we require $cost_{\mathcal{A}}(r(d)) \leq \Delta$.

Example 2: In this paper we consider as resource Δ the time available to the adversary for breaking the key (e.g., the key deployment time), which the adversary can spend between sequentially making m side channel measurements (each of which we assume take time τ_{on}) and n search steps (each of which we assume take take time τ_{off}). The adversary's actions are hence

$$A = \{(m, n) \mid m\tau_{off} + n\tau_{on} \leq \Delta\} \ .$$

Note that this definition of A also caters for parallel attacks by considering more general notions of cost and resources, which we forgo for the sake of concreteness.

D. Solving the Game

We use the standard solution concept for multi-stage games for characterizing optimal countermeasure configurations. Namely, we use subgame perfect equilibrium (see e.g. [14]), which is a combination of strategies (d^*, r^*) of the two players such that none of the players can obtain a higher payoff by deviating from the strategy, if the other player sticks to their strategy.

Definition 3: A *subgame perfect (Nash) equilibrium (SPE)* of a game $G = [(\mathcal{D}, \mathcal{A}), (D, A), (u_{\mathcal{D}}, u_{\mathcal{A}})]$ is a defender's action d^* and an adversary's response function r^*, such that

(i) for all $d \in D$: $u_{\mathcal{D}}(d, r^*(d)) \leq u_{\mathcal{D}}(d^*, r^*(d^*))$, and

(ii) for all r: $u_{\mathcal{A}}(d^*, r(d^*)) \leq u_{\mathcal{A}}(d^*, r^*(d^*))$.

The SPEs of a game can be obtained by *backward induction*, which consists of solving two optimization problems: First, we find the adversary's response function that gives an optimal response to each of the defender's actions. Then, taking into account the optimal response function, we find the optimal defense. The existence of an equilibrium is guaranteed by the following theorem.

Theorem 1 ([12], [26], [35]): A finite game of perfect information has a pure-strategy SPE. This SPE can be computed in time $O(|D| \cdot |A|)$, under the assumption that $u_{\mathcal{D}}$ and $u_{\mathcal{A}}$ can be evaluated in time $O(1)$.

Here, *finite* refers to the number of players' actions and number of stages in the game, *perfect information* refers to the fact that the second player knows which action the first player has done, and *pure-strategy* refers to the players' strategies being deterministic.

E. Soundness of Solutions Based on Probability Bounds

The definition of the countermeasure configuration game, and hence its solution, rely on the probability p of a breach. When solving the game for practical systems, we need to resort to approximations of p. Specifically, in this paper we work with upper bounds $\hat{p}$ on p which we introduce in Section III.

However, using upper bounds may lead to unsound decisions. For example, over-approximating p by $\hat{p} = 1$ in the utility function defined in (2) may suggest not to deploy any countermeasures. That is, the solution of the game based on $\hat{p}$ will leave the system vulnerable, while the solution based on p may command the deployment of a countermeasure. We now show that the countermeasure configuration game yields sound solutions when used with bounds on the probability of breach, in the sense that one errs on the secure side when solving the game using $\hat{p}$ instead of p.

For this, consider countermeasure configuration games G and $\hat{G}$ with $u_{\mathcal{A}}(\cdot) = p(\cdot)$, $\hat{u}_{\mathcal{A}}(\cdot) = \hat{p}(\cdot)$, and $u_{\mathcal{D}} = \hat{u}_{\mathcal{D}}$, such that $\hat{G}$ uses an over-approximation of the probability of breach in the sense that $\hat{p} \geq p$ pointwise. The following proposition shows that if a defender chooses an optimal defense strategy with respect to $\hat{G}$, and uses it to play G, the probability of breach will still be upper bounded by $p_{\max}$.

Proposition 1: Let d^*, r^* be a SPE of $\hat{G}$. Then, for all r,

$$p(d^*, r(d^*)) \leq p_{\max}$$

Proof: For all adversary strategies r, we have

$$p(d^*, r(d^*)) \overset{(1)}{\leq} \hat{p}(d^*, r(d^*)) \overset{(2)}{\leq} \hat{p}(d^*, r^*(d^*)) \ ,$$

where (1) follows from $p \leq \hat{p}$; and (2) holds because $r^* = \arg\max_r \hat{p}(d, r(d))$, i.e., for each r, d, $\hat{p}(d, r(d)) \leq \hat{p}(d, r^*(d))$. As d^* is the defender's optimal action in $\hat{G}$, $\hat{p}(d^*, r(d^*)) \leq p_{\max}$, from which the assertion follows. ■

Proposition 1 shows that the countermeasure configuration game leads to sound solutions when used with bounds on the probability of breach, whereas we have seen that the utility defined by (2) does not. We leave a general characterization of these two kinds of utilities to future work.

III. Bounds on the Probability of Key Recovery

A. Our Approach

In this section, we derive bounds for the probability of recovering the secret key of a public-key cryptosystem in a combined timing/algebraic attack. Our bounds are practical enough to justify the choice of a leaky, non-constant time implementation over a defensive constant-time implementation for standard ElGamal in some settings, see Section V. We derive our bounds in the generic group model and under the assumption that the cryptosystem is protected against timing attacks by an idealized form of input blinding.

The starting point for our proof is the bound discussed in Section II-A of the amount of information contained in n execution time measurements of a blinded implementation of modular exponentiation. The technical challenge we face is to turn this bound into a guarantee against an adversary that can mount a combined timing/algebraic attack: Existing notions of leakage (e.g. [4], [33]) do not easily combine with the adversary models used in cryptography, and cryptographic notions of security (e.g. [19]) do not cater for this weak yet realistic leakage model.

We identify unpredictability entropy [17] as a suitable notion of entropy for extending leakage bounds to notions of hardness w.r.t. computational adversaries. Unpredictability entropy is versatile enough to accommodate for different classes of adversaries and it satisfies a chain rule, which provides an interface to the leakage bounds.

There are few facts known about the unpredictability entropy of computational problems [3], nor are there generally accepted hardness assumptions.[1] To compensate for this lack, we cast Shoup's lower bound for computing discrete logs in generic groups in terms of the unpredictability entropy w.r.t. an adversary who can perform m group operations. This step requires a slight adaptation of unpredictability entropy and the chain rule to the generic group model. We then connect the leakage bounds with the entropy bounds for the discrete log to obtain the desired bound (in terms of n and m) for combined side channel/algebraic adversaries.

[1]It is known that RSA private keys have low unpredictability entropy because they can be efficiently recovered if a small fraction of the key bits are known [16].

B. Generic Algorithms for Computing Discrete Logarithms

A generic group algorithm is an algorithm that solves problems over groups by only performing group operations and equality tests. That is, it does not make any use of the specific representation of group elements. Generic algorithms are an attractive model of computation for security because one can use them to establish lower bounds for computational problems [32]; their disadvantage is that they may underrate the power of real adversaries.

We next introduce the notion of generic algorithms for computing discrete logarithms in cyclic groups of order k, which are isomorphic to the additive group $\mathbb{Z}_k$. We use the definition by Maurer [28].

Definition 4 (Generic algorithm in groups): A *generic algorithm* $\mathsf{A}^m_{\mathcal{G}(X)}$ over $\mathbb{Z}_k$ is an algorithm which can make m computation steps, each consisting of one query to a group oracle $\mathcal{G}$. The oracle has an internal state $v_0, v_1, v_2, \ldots, v_m$, where $v_0 = x$ is initialized with a secret sampled from a random variable X with $ran(X) = \mathbb{Z}_k$. The t-th query to the oracle consists of one of the following two operations:

1) constant insertion: algorithm inputs $c \in \mathbb{Z}_k$, oracle sets $v_t := c$
2) variable addition: algorithm inputs $i, j < t$, oracle sets $v_t := v_i + v_j \pmod{k}$

The oracle then outputs the results of testing v_t for equality with all other elements in the internal state, i.e. $(v_t = v_i)_{i\in\{0,\ldots,t-1\}}$.

In this model, each v_i can be represented as a polynomial $v_i = ax+b$, where a and b are known to the adversary. This is because each v_i is derived from x and the inserted constants by repeated addition. If $v_i = v_j$ for $v_i = ax + b$ and $v_j = a'x+b'$ with $(a, b) \neq (a', b')$, we say that there is a *collision* between v_i and v_j. For the special case of a group of prime order q, a collision can be used for recovering x, because the equation $(a - a')x + (b - b') = 0$ has a unique solution in $\mathbb{Z}_q$.

In m queries, an adversary can establish at most $\binom{m}{2}$ equations of the form $v_i = v_j$, each of which has a unique solution. The probability of hitting one of them when sampling uniformly at random from $\mathbb{Z}_k$ is upper-bounded by $\binom{m}{2}/q$. Note that collisions are the only way for recovering x in this model, i.e. the bound holds for *any* generic algorithm for extracting x from the oracle.

The bound extends to bounds for cyclic groups of arbitrary order k via the Chinese remainder theorem.

Theorem 2 ([28], [32]): Let $\mathsf{A}_{\mathcal{G}}^m$ be a generic algorithm over $\mathbb{Z}_k$, q the largest prime factor of k, and X uniformly distributed. Then

$$P[\mathsf{A}_{\mathcal{G}(X)}^m = X] \leq \frac{m^2}{q} \ .$$

C. Unpredictability Entropy

Unpredictability entropy [17] is a notion of entropy that generalizes conditional min-entropy to computational settings. More specifically, it captures the probability of a resource-bounded algorithm estimating the value of a random variable from that of another. Unpredictability entropy has been defined for different computational models, such as circuits of bounded size [17] and polynomial time algorithms [3]. Here we define unpredictability entropy w.r.t. bounds on the number of accesses to an oracle $\mathcal{O}(X)$ that receives as input the value of a random variable X.

Definition 5 (Unpredictability entropy): We say that X has *unpredictability entropy at least t bits* for m calls to $\mathcal{O}$, written $H_{\mathcal{O}}^m(X) \geq t$, if for all algorithms which can make at most m queries to $\mathcal{O}(X)$,

$$P[\mathsf{A}_{\mathcal{O}(X)}^m = X] \leq 2^{-t} \ .$$

Example 3: Let p be prime, g a generator of $\mathbb{Z}_p^*$ and q the largest prime factor of $p-1$. The problem of computing the discrete logarithm x of g^x has unpredictability entropy of least $\log_2 \frac{q}{m^2}$ for m calls to $\mathcal{G}$. This is because the multiplicative group $\mathbb{Z}_p^*$ is cyclic, and hence isomorphic to the additive group $\mathbb{Z}_{p-1}$. Then Theorem 2 applies.

Example 4: For $m = 0$, the best algorithm for predicting X is one that has the most likely value of X hardcoded, i.e. unpredictability entropy and min-entropy coincide:

$$H_{\mathcal{O}}^0(X) = H_\infty(X) \ .$$

Next we define a conditional version of unpredictability entropy. For this we consider an algorithm $\mathsf{A}_{\mathcal{O}(X)}^m(Y)$ that can observe the output of a random variable Y that is jointly distributed with the oracle input X.

Definition 6 (Conditional Unpredictability Entropy): Let X, Y be random variables. We say that X has *unpredictability entropy at least t conditioned on Y* with respect to oracle $\mathcal{O}$, written $H_{\mathcal{O}}^m(X|Y) \geq t$, if for all algorithms that can take one sample from Y and can make at most m queries to $\mathcal{O}(X)$,

$$P[\mathsf{A}_{\mathcal{O}(X)}^m(Y) = X] \leq 2^{-t} \ .$$

The following chain rule [25] shows that unpredictability entropy decreases gracefully when conditioned on additional information. Note that this is not necessarily true for other notions of computational entropy, such as HILL entropy [25].

Lemma 1 (Chain rule): For random variables X, Y with $\log_2 |ran(Y)| = \ell$,

$$H_{\mathcal{O}}^m(X) \geq t \Rightarrow H_{\mathcal{O}}^m(X|Y) \geq t - \ell \ .$$

Proof: Assume that $H_{\mathcal{O}}^m(X|Y) < t-\ell$. By the definition of unpredictability entropy it follows that there exists an algorithm $\mathsf{A}_{\mathcal{O}}^m$ such that $P[\mathsf{A}_{\mathcal{O}(X)}^m(Y) = X] > 2^{-t+\ell}$. We construct the algorithm $\mathsf{B}_{\mathcal{O}}^m$ which takes no input, chooses $y' \in ran(Y)$ u.a.r. (which we represent by the random variable Y'), and returns $\mathsf{A}_{\mathcal{O}(X)}^m(y')$. Then we obtain

$$\begin{aligned} P[\mathsf{B}_{\mathcal{O}(X)}^m = X] &= P[\mathsf{A}_{\mathcal{O}(X)}^m(Y') = X] && (3)\\ &= P[\mathsf{A}_{\mathcal{O}(X)}^m(Y) = X \wedge Y = Y'] && (4)\\ &= P[\mathsf{A}_{\mathcal{O}(X)}^m(Y) = X] \cdot P[Y = Y'] && (5)\\ &= P[\mathsf{A}_{\mathcal{O}(X)}^m(Y) = X] \cdot 2^{-\ell} && (6)\\ &> 2^{-t+\ell} \cdot 2^{-\ell} = 2^{-t} \ , && (7) \end{aligned}$$

where (5) follows because both events are independent, i.e., the fact that Y collides with a uniformly chosen element does not affect the probability that the algorithm guesses X correctly, and vice versa. ■

D. Blinded Side Channels

We briefly revisit the notion of leakage that captures timing side channels of cryptographic algorithms with input blinding and bucketing as a countermeasure, see Section II-A. The assumption is that the execution time depends only on the key (which remains fixed over multiple executions) and the blinded message (which is chosen randomly and independently in each execution). The model abstracts from potential timing leaks through the blinding/unblinding operations and system state such as caches.

A *blinded channel* for X is a family of random variables $\{O_x\}_{x \in ran(X)}$, one for every x, with shared range $B = ran(O_x)$ of bounded size $b = |B|$. For a fixed secret x, making n timing measurements corresponds to taking n independent samples from O_x. As the samples are independent, their relative ordering does not contain information about x; the information about x contained in these samples can hence be represented by the *type* Y_n of the sequence, i.e. the vector of relative frequencies [23]. The number of such vectors (and hence the information about X contained in n timing measurements of a blinded channel) is bounded as follows.

Theorem 3: $|ran(Y_n)| \leq (n + 1)^{b-1}$

Theorem 3 yields an upper bound for the amount of information contained in n timing measurements. It can be improved slightly using a more careful counting argument [24], and more significantly by using the fact that timing observations are not uniformly distributed but rather follow a multinomial distribution with b possible outcomes. We opt for the simple bound of Theorem 3 because it is tight enough for our purposes and has the advantage of a polynomial expression.

E. Bounds for Combined Adversaries

We now leverage the results presented in this section to derive bounds on the probability of key recovery by a combined algebraic/side channel attack. In particular, we use the chain rule (Lemma 1) to combine the lower bounds for the unpredictability entropy (Theorem 2 and Example 3) with the upper bounds on the leakage (Theorem 3) and obtain the following result.

Theorem 4 (Generic algorithms with side channels): Let $\mathsf{A}^m_{\mathcal{G}}$ be an algorithm that can make m calls to a group oracle and n measurements of a blinded channel. Then

$$P[\mathsf{A}^m_{\mathcal{G}(X)}(Y_n) = X] \leq \frac{m^2(n+1)^{b-1}}{q} \ ,$$

where parameters b and q denote the range of the blinded channel and the largest prime divisor of the group order, respectively.

Our modeling captures the case in which the adversary first performs n timing measurements and then performs m calls to the group oracle. However, the bounds we derive also hold for adversaries that can interleave timing measurements and oracle queries. The reason for this is that the adversary cannot influence the timing measurements, which is why nothing is gained by postponing a timing measurement until after an oracle query.

IV. Computing the Equilibrium

In this section we show how to solve the countermeasure configuration game for a discrete logarithm based cryptosystem that is protected with blinding and bucketing. As described in Section II-D, a pure-strategy equilibrium of a generic two-stage game can be computed by backward induction, that is, sequentially solving two optimization problems. We describe both optimization problems below.

For this, we rely on notation introduced in Examples 1 and 2 and the bounds derived in Section III. For their connection, consider a cyclic group $\mathbb{Z}^*_p$, where p is prime and q is the largest prime factor of $p-1$, as in Example 3. We use the bit lengths $(|p|, |q|)$ to instantiate the abstract notion of key length k in Section II. In particular, we use $(|p|, |q|)$ to describe desired properties of the modulus, but without referring to specific p and q.

A. Adversary's Optimization Problem

The adversary's optimization problem is to find a response function r^* that maps defender actions to an adversary action that maximizes the utility $u_{\mathfrak{A}} = p(\cdot)$. That is, for a defender's choice of key length $k = (|p|, |q|)$ and bucketing b, we need to compute the numbers m^* and n^* of offline and online steps, respectively, that maximize $p((k, b), (m, n))$ subject to the resource constraint $m\tau_{off} + n\tau_{on} \leq \Delta$.

For the solution, first observe that the adversary's utility (the probability of breach) is maximized when all resources Δ are invested in the attack; thus we treat the resource constraint as an equality, and use it to express m in the bound from Theorem 4:

$$p((k, b), (m, n)) \leq \frac{1}{\tau^2_{off}} \frac{(\Delta - n\tau_{on})^2(n+1)^{b-1}}{2^{|q|}} \ . \qquad (8)$$

The case $b = 1$ corresponds to a constant-time implementation, which is why the utility is maximized when time is spent on offline guessing rather than on online queries, i.e., $n^* = 0$ and $m^* = \Delta/\tau_{off}$.

The case $b \geq 2$ corresponds to an implementation with timing leaks. We symbolically compute the maximum utility by solving for n in $\frac{\partial}{\partial n}(\cdot) = 0$ applied to (8). This yields the maximum utility at $n^* = (\Delta(b-1) - 2\tau_{on})/((b+1)\cdot\tau_{on})$ and the adversary response

$$r^*(k, b) = ((\Delta - n^*\tau_{on})/\tau_{off}, n^*) \ .$$

B. Defender's Optimization Problem

The defender's optimization problem is to identify the defense d^* that maximizes the defender's utility $-cost_{\mathfrak{D}}(d^*)$, where the adversary response r^* is given. Technically, solving this problem requires computing a bucketing b^* and a key length k^* that minimize the average execution time $cost_{\mathfrak{D}}$ subject to the constraint $p((k^*, b^*), r^*(k^*, b^*)) \leq p_{\max}$.

Here we face the challenge that there are no analytical models describing $cost_{\mathfrak{D}}$ as a function of key length and number of buckets. Instead, we instantiate each value of $cost_{\mathfrak{D}}$ by empirical analysis of the corresponding implementation. We propose a simple heuristic to guide the search and avoid the evaluation of $cost_{\mathfrak{D}}$ on too many parameters.

Restricting the search space: We instantiate $p_{\max}$ as the probability of breach within time Δ of a key of reference length k_{ref} by an adversary that has no access to timing information. The rationale behind using such adversaries as a baseline is that we can rely on standards such as key length recommendations [2] for justified parameter values.

We then use (8) and the adversary's response n^* (as a function of (k, b)) to restrict the parameter space to the (k, b) that satisfy:

$$\frac{(\Delta - n^*\tau_{on})^2(n^*+1)^{b-1}}{\tau^2_{off}\ 2^{|q|}} \leq \frac{\Delta^2}{\tau^2_{off} 2^{|q|_{ref}}} = p_{\max} \qquad (9)$$

Empirically finding the optimum: The empirical evaluation of $cost_{\mathfrak{D}}(k, b)$ of a real implementation is expensive. To avoid evaluation on the entire parameter space defined by (9) we use a simple heuristic based on a weak assumption about the implementation, namely, that the average execution time grows with the length of the key.

With this, we explore the search space as follows. For $p_{\max}$ fixed and $b = 1, 2, \ldots$ we define k_b to be the smallest key that achieves security $p_{\max}$, i.e.

$$k_b = \min\{k \mid p((k, b), r^*(k, b)) \leq p_{\max}\}$$

As the execution time grows with the key length we do not need to consider keys beyond k_b because they will have less utility to the defender. For $b = 1, 2, \ldots$ we empirically determine the distribution of execution times of an implementation with keys of length k_b, compute the optimal bucket boundaries into b buckets using dynamic programming [23], and define $cost_{\mathfrak{D}}(k_b, b)$ as the corresponding average execution time.

In theory, the number of buckets that need to be considered is upper bounded by the number of possible execution times. In practice (see Section V), the shape of $cost_{\mathfrak{D}}(k_b, b)$ becomes apparent after inspection of small values, which allows identifying (k^*, b^*).

V. Case Study

In this section, we report on a case study where we identify the optimal configuration of countermeasures against timing attacks on ElGamal decryption, following the approach developed in Sections II, III, and IV.

A. Experimental Setup

We analyze the ElGamal implementation of libgcrypt 1.6.1, compiled with GCC 4.8.2 on Ubuntu 14.04. We measure execution time in terms of the number of executed CPU instructions instead of real time, thereby abstracting from the influence of the microarchitecture. We determine the distribution of execution times corresponding to particular key lengths and countermeasure configurations by sampling the time for decrypting 10^5 random ciphertexts with different keys. As a benchmarking tool, we rely on the PAPI library [30].

B. ElGamal Implementation in Libgcrypt

In libgcrypt, ElGamal is implemented over the multiplicative group $\mathbb{Z}_p^*$. For deriving bounds on the probability of breach using Theorem 4, we need to know the size of the largest prime factor q of $p - 1$. In libgcrypt, lower bounds for q can be directly read off the source code: the key-generation algorithm ensures that q has a bit-length of at least `qbits`, which is chosen according to a pre-defined table (see Figure 1). The secret exponent x is then taken of size `qbits` plus a safety margin, which results in faster decryption[2].

ElGamal decryption in libgcrypt is performed in three steps: modular exponentiation (we chose the square-and-multiply option), inversion, and multiplication. Blinding is not readily available in libgcrypt; we implement it relying on pre-computed randomness (see, e.g., [20]), which adds the overhead of two multiplications to the decryption time.

$\lvert p \rvert$	1024	1536	2048	2560	3072
`qbits`	165	198	255	249	269

Fig. 1: Excerpt from "Wiener's table", used in libgcrypt for determining the minimal bit-length `qbits` of p's factors.

C. Constant-Time ElGamal

We modify the libgcrypt source code to estimate the timing of a constant-time implementation of ElGamal. The modifications include always performing multiplication in the square-and-multiply exponentiation ($\approx$ 35% overhead), forcing multi-precision integer comparison to always iterate over the entire numbers ($\approx$ 60% overhead), as well as performing dummy operations to even out the timing of conditional branches in the routines for squaring, multiplication, and division ($\approx$ 4% overhead). Additionally, compiler optimizations were switched off ($\approx$ 10% overhead). In total, the overhead of the performed changes is $\approx$ 125 to 155%, depending on the modulus size.

To eliminate timing variations in modular inversion, we replace libgcrypt's implementation with a straightforward application of Fermat's little theorem ($a^{-1} = a^{p-2} \mod p$), where we rely on the constant-time exponentiation. Because the slowdown using this approach is significant compared to state-of-the-art algorithms for modular inversion [10], for fairness of our analysis we use the Fermat-based inversion for both constant-time and non-constant-time timing measurement.

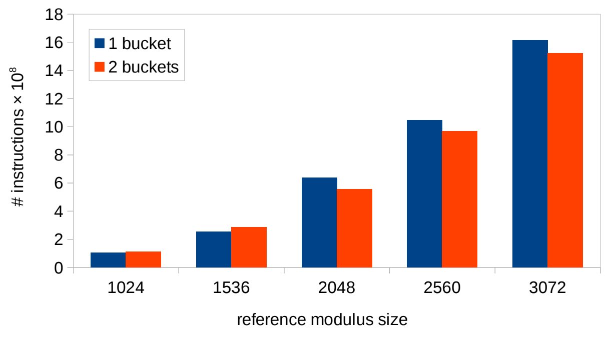
D. Results

In our experiments we analyze the influence of the following parameters on the optimal countermeasure configuration: reference modulus size $|p|_{ref}$ as specified by k_{ref}; the key deployment time Δ; the key generation algorithm; and the access rate $\rho_{acc} = \tau_{on}^{-1}$. The latter parameter gives the number of accesses the adversary can make to timing observations per second, and can be influenced e.g. by limiting the rate of server requests, or by using denial-of-service preventions. In practice, the choice of a key size affects decryption time, and thus also the timing of an offline step τ_{off}. We avoid explicit instantiation of τ_{off} in dependence of the key size by over-approximating the adversary's capabilities, setting the value of τ_{off} with a key of size k to be τ_{off} with the corresponding k_{ref}, which gives a sound solution according to Proposition 1.

1) Varying the Modulus Size: We first consider varying the modulus size of keys generated with the libgcrypt default key generation algorithm, for fixed $\Delta = 365$ days and $\rho_{acc} = 100$ accesses per second. As depicted in Figure 2, for modulus sizes $|p|_{ref} \in \{1024, 1536\}$, the optimal defense is to use a constant-time implementation,

[2]In libgcrypt's source code, this is explained vividly: "I don't see a reason to have a x of about the same size as the p. It should be sufficient to have one about the size of q or the later used k plus a large safety margin. Decryption will be much faster with such an x."

i.e., $d^* = (k_{ref}, 1)$. For bigger modulus sizes, we obtain the optimum at a non-constant-time implementation with $b = 2$ buckets; the corresponding optimal modulus sizes are depicted in Figure 2b.



(a) Average cost (in number of CPU instructions).

b						
1	$\lvert p\rvert_{ref}$	1024	1536	2048	2560	3072
2	$\lvert p\rvert$	1478	2087	2683	3323	3898

(b) Modulus sizes $|p|$ (in bits) providing the same protection with 2-bucketing as modulus sizes $|p|_{ref}$ with one bucket.

Fig. 2: Varying the modulus sizes for default libgcrypt keys. The results are given for $\Delta = 365$ days, $\rho_{acc} = 100$ accesses per second.

2) Varying the Access Rate ρ_{acc}: Increasing the access rate ρ_{acc} gives the adversary more possibilities to collect timing observations; thus, a defender deploying a non-constant time implementation needs to increase the modulus size to compensate for this information loss. In Figure 3 we demonstrate that this can increase the cost of the non-constant time implementation enough for the optimum to shift from $b = 2$ to $b = 1$, i.e., the defender will prefer a constant-time implementation.

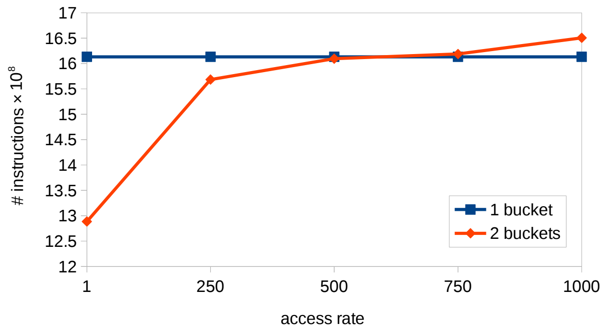


Fig. 3: Average cost (in number of CPU instructions) for varying access rate ρ_{acc} (in accesses per second). The results are given for $|p|_{ref} = 3072$ bit, $\Delta = 365$ days.

3) Varying the Key Deployment Time: Varying the deployment time has a similar effect as varying the access rate: an adversary has more time to collect timing observations. For example, if the deployment time is decreased, the defender's preference may shift from a constant to a non-constant-time implementation, as depicted in Figure 4.

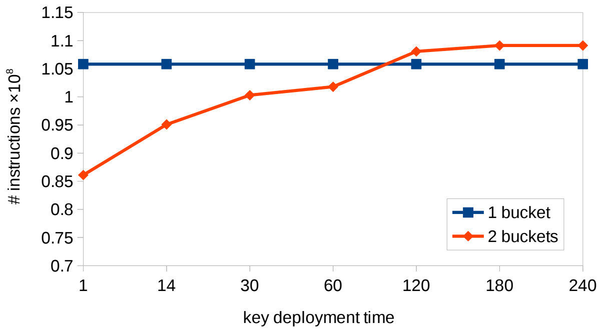


Fig. 4: Average cost (in number of CPU instructions) for varying deployment time Δ (in days). The results are given for $|p|_{ref} = 1024$ bits, $\rho_{acc} = 100$ accesses per second.

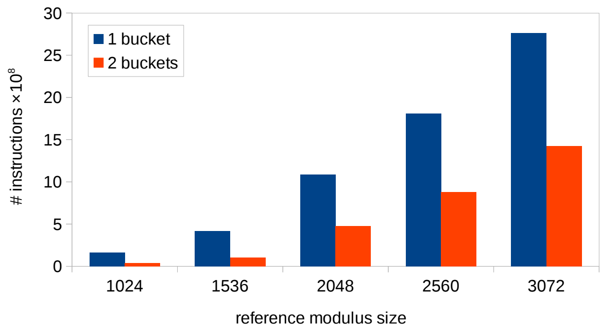
4) Using a Safe Prime Modulus: An alternative approach for key generation makes sure that the group modulus p is a *safe prime*, i.e., $p = 2q + 1$ for a prime q. For example, the pycrypto library uses Algorithm 4.86 in [29] to generate p as a safe prime. As a result, q is guaranteed to have a bit length of $|p| - 1$.

Figure 5 illustrates the effect of varying the bit-size of the safe prime p, for fixed $\Delta = 365$ days and $\rho_{acc} = 100$ accesses per second. A comparison with Figure 2 shows that the benefits of using a non-constant time implementation are more substantial for safe primes than for default libgcrypt keys. The reason is that, for the same bit-length, safe primes provide more security in terms of Theorem 2, which is why the information loss from side-channel observations in a non-constant time implementation can be compensated by adding fewer bits to the key.

5) Varying the Number of Buckets: When increasing b, the defender has to increase the corresponding modulus size in order to ensure that the desired security level is met; for this to be economically feasible, the overhead from the bigger key needs to be compensated by savings from the countermeasure. In all cases we consider, increasing b above 2 did not fulfill this requirement, and thus all obtained optima were for $b = 1$ or $b = 2$. This is the case even in cases where an increase in b requires only a small increase in $|p|_{ref}$, as is the case with safe primes (see Figure 6).

E. Use Cases

In the following we choose the parameters to reflect two example use cases. We set the $|p|_{ref} = 2048$, which is the default *GnuPG* setting.

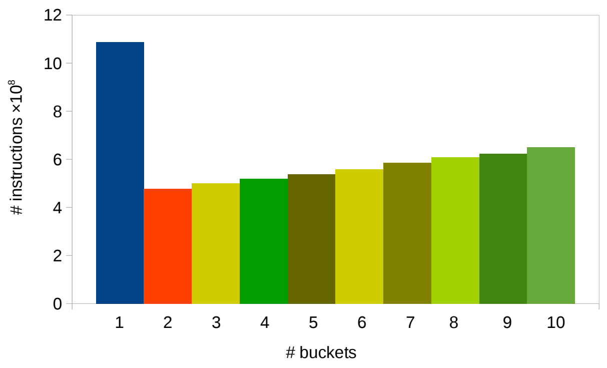


(a) Average cost (in number of CPU instructions).

b						
1	$\|p\|_{ref}$	1024	1536	2048	2560	3072
2	$\|p\|$	1478	2087	2683	3323	3898

(b) Modulus sizes $|p|$ (in bits) providing the same protection with 2-bucketing as modulus sizes $|p|_{ref}$ with one bucket.

Fig. 5: Varying the modulus sizes for safe primes. The results are given for $\Delta = 365$ days and $\rho_{acc} = 100$ accesses per second.



(a) Average cost (in number of CPU instructions).

buckets	1	2	3	4	5	6
$\|p\|$	2048	2078	2109	2139	2170	2201

(b) Modulus sizes $|p|$ (in bits) providing the same protection for varying number of buckets.

Fig. 6: Varying the number of buckets, with safe prime modulus of reference size $|p|_{ref} = 2048$. The results are given for $\rho_{acc} = 100$ accesses per second and time of deployment $\Delta = 365$ days.

As a first use case, we consider a proxy server that decrypts incoming emails. In this scenario, the access rate ρ_{acc} can be expected to be small (we set it to $\rho_{acc} = 10$), while key deployment times can be expected to be higher. Figure 7 depicts that in this scenario, even for very long periods of time, we obtain the optimum configuration with the non-constant-time implementation ($b = 2$). Compared to the constant-time implementation, the optimal configurations give savings between 23% and 30%.

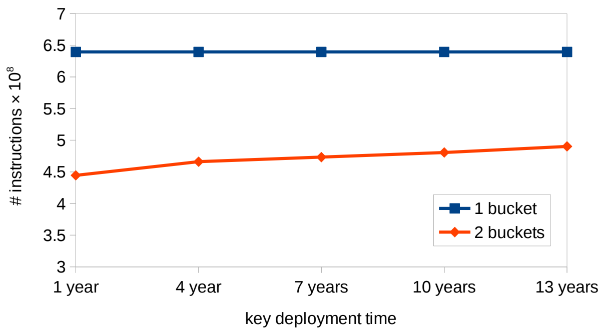


Fig. 7: Average cost (in number of CPU instructions) for varying deployment time Δ. The results are given for $|p|_{ref} = 2048$ bit, $\rho_{acc} = 10$ accesses per second.

As a second use case, we consider an Internet-facing server which handles user requests. In this scenario, keys may have a shorter life-time, which we set to $\Delta = 90$ days; however, a higher access rate translates to a higher throughput, which may be a critical goal. For this scenario, Figure 8 shows that the non-constant-time implementation is the optimal solution even for large access rates; the expected savings compared to the constant-time implementation are between 10% and 28%.

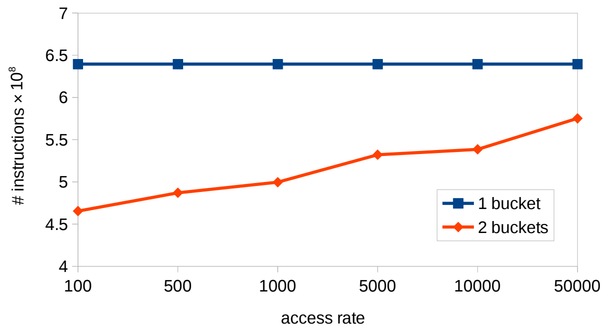


Fig. 8: Average cost (in number of CPU instructions) for varying access rate ρ_{acc} (in accesses per second). The results are given for $|p|_{ref} = 2048$ bit, $\Delta = 90$ days.

VI. Related Work

Game theory and security trade-offs: A substantial body of research uses game theory for reasoning about trade-offs between security (or privacy) and conflicting goals. Stackelberg games are particularly prevalent in the literature, and the fact that the adversary reacts to the defender's commitment reflects Kerckhoff's principle.

For example, [34] uses Stackelberg games for computing the optimal placement of checkpoints and canine patrol routes for achieving (physical) security, e.g., at airports. In that setting, [6] considers a defender who aims to minimize cost while maintaining a fixed level of protection, which is similar in spirit to our definitions of security. In [7], the authors use Stackelberg games to reason about the configuration of audit mechanisms, where the trade-off is between the cost of detecting or preventing incidents. In [31] the authors use Stackelberg games to reason about the configuration of location privacy mechanisms, where the trade-off is between privacy and service quality.

All of the above approaches consider probabilistic (i.e. mixed) strategies. In contrast, we consider deterministic (i.e. pure) strategies, which is why the equilibrium can be computed by enumeration [12]. However, we deviate from this algorithm because the set of adversary's actions is too large and the defender's utility is expensive to evaluate.

Work in rational cryptography (see [15] for a recent overview) also considers adversaries as players aiming to maximize their utilities. One advantage of this adversary model is that one can circumvent impossibility results that hold for stronger, worst-case adversaries. The countermeasure configuration game is atypical in the sense that it captures worst-case adversaries who spend all resources and relies on the utility function to describe their optimal usage.

Quantitative information flow analysis: The bounds for the countermeasure configuration game we presented rely on quantitative information-flow analyses that account for the aggregate information leaked in multiple executions. In our case study, we relied on approaches that deal with blinded inputs [9], [23]. Approaches that can deal with adversarially chosen input input [8], [22] are currently limited to systems with small state-spaces due to the lack of efficient abstractions. With progress on such abstractions, bounds such as the ones from Theorem 3 could be obtained automatically from code and platform models [13].

g-vulnerability [4] is a notion of entropy that accounts for general notions of adversary's gain. While it is possible to cast the generic discrete logarithm problem in terms of a specific g-function (the adversary gains 1 with a collision and 0 otherwise), we chose to rely on unpredictability entropy as a notion because it explicitly models resource-bounded computation and hence provides a natural connection to a variety of adversary models in cryptography.

Mardziel et al. [27] consider information-flow in dynamic systems, where defender and adversary can interact. Their focus is on secrets that are dynamically changing, whereas our approach specifically considers the aggregation of information about long-term secrets, such as secret keys.

Zhang et al. [36] study quantitative approaches for mitigating timing leaks by adaptively delaying outgoing messages. They consider a covert channel adversary, i.e. one that aims to transmit information from within a system. This adversary is stronger than, and the corresponding notion of success (i.e. successful transmission) is different from, the ones we consider in this paper. It would be interesting to see how their approach can be cast into a decision-theoretic context.

Leakage-resilient cryptography: Belaïd et al. [5] also reason about trade-offs between security and performance in side-channel attacks, where they focus on the decision between the masking countermeasure and a leakage-resilient primitive. As in our work, they investigate which implementation offers the best performance for a fixed level of security. They consider power analysis attacks and use the best known attacks as a security benchmark, whereas we consider timing attacks and use the best known security guarantees as a benchmark. Our approach relies on game theory for framing and solving the decision problem, which offers the advantage of a clean interface between the security guarantees and the algorithmic challenges.

Kiltz and Pietrzak [19] present a leakage-resilient variant of ElGamal, based on multiplicative secret sharing. They consider a more general leakage model and prove indistinguishability under chosen cipher attack (also in the generic group model), whereas we only prove security against key recovery attacks. The advantage of aiming for weaker guarantees is that they apply to *standard* ElGamal and that their simplicity makes them easily applicable in a game-theoretic context.

The connection of timing leakage of blinded implementations to cryptographic security has been studied in [24], for asymptotic notions of security. In contrast, the bounds we develop in this paper are concrete, which is required for using them in the context of the countermeasure configuration game.

VII. Conclusions and Future Work

We have presented a systematic approach for determining the optimal protection against timing attacks, where we make use of a number of simple but powerful tools from game theory, information theory, and cryptography. The results we obtain are rigorous but practical enough to justify the use of a fast but leaky implementation of ElGamal over a defensive constant-time implementation.

Acknowledgements: This work was partially funded by Spanish Project TIN2012-39391-C04-01 StrongSoft and Madrid Regional Project S2013/ICE-2731 N-GREENS.

References

[1] Functional safety of electrical, electronic and programmable electronic safety related systems–IEC 61508. www.iec.ch/functionalsafety/.

[2] ECRYPT II Yearly Report on Algorithms and Key Lengths (2011), June 2011.

[3] D. Aggarwal and U. Maurer. The leakage-resilience limit of a computational problem is equal to its unpredictability entropy. In *ASIACRYPT*, pages 686–701. Springer, 2011.

[4] M. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith. Measuring information leakage using generalized gain functions. In *CSF*, pages 265–279. IEEE, 2012.

[5] S. Belaïd, V. Grosso, and F.-X. Standaert. Masking and leakage-resilient primitives: One, the other(s) or both? *IACR Cryptology ePrint Archive*, 2014:53, 2014.

[6] S. Bhattacharya, V. Conitzer, and K. Munagala. Approximation algorithm for security games with costly resources. In *Internet and Network Economics*, pages 13–24. Springer, 2011.

[7] J. Blocki, N. Christin, A. Datta, A. D. Procaccia, and A. Sinha. Audit games. In *IJCAI*, pages 41–47. AAAI Press, 2013.

[8] M. Boreale and F. Pampaloni. Quantitative multirun security under active adversaries. In *QEST*, pages 158–167. IEEE, 2012.

[9] M. Boreale, F. Pampaloni, and M. Paolini. Asymptotic information leakage under one-try attacks. In *Foundations of Software Science and Computational Structures*, pages 396–410. Springer, 2011.

[10] J. W. Bos. Constant time modular inversion. *Journal of Cryptographic Engineering*, 4(4):275–281, 2014.

[11] D. Brumley and D. Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[12] V. Conitzer and T. Sandholm. Computing the optimal strategy to commit to. In *EC*, pages 82–90. ACM, 2006.

[13] G. Doychev, D. Feld, B. Köpf, L. Mauborgne, and J. Reineke. CacheAudit: A Tool for the Static Analysis of Cache Side Channels. In *USENIX Security Symposium*. USENIX, 2013.

[14] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

[15] J. A. Garay, J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. In *FOCS*, pages 648–657, 2013.

[16] N. Heninger and H. Shacham. Reconstructing rsa private keys from random key bits. In *CRYPTO*, pages 1–17. Springer, 2009.

[17] C.-Y. Hsiao, C.-J. Lu, and L. Reyzin. Conditional computational entropy, or toward separating pseudoentropy from compressibility. In *EUROCRYPT*, pages 169–186. Springer, 2007.

[18] S. Jana and V. Shmatikov. Memento: Learning secrets from process footprints. In *SSP*, pages 143–157. IEEE, 2012.

[19] E. Kiltz and K. Pietrzak. Leakage resilient elgamal encryption. In *ASIACRYPT*. Springer, 2010.

[20] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO*, pages 104–113. Springer, 1996.

[21] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, pages 388–397. Springer, 1999.

[22] B. Köpf and D. Basin. An Information-Theoretic Model for Adaptive Side-Channel Attacks. In *CCS*, pages 286–296. ACM, 2007.

[23] B. Köpf and M. Dürmuth. A provably secure and efficient countermeasure against timing attacks. In *CSF*, pages 324–335. IEEE, 2009.

[24] B. Köpf and G. Smith. Vulnerability bounds and leakage resilience of blinded cryptography under timing attacks. In *CSF*, pages 44–56. IEEE, 2010.

[25] S. Krenn, K. Pietrzak, A. Campus, A. Wadia, and D. Wichs. A counterexample to the chain rule for conditional hill entropy. 2014.

[26] H. W. Kuhn. Extensive games and the problem of information. *Contributions to the Theory of Games*, 2(28):193–216, 1953.

[27] P. Mardziel, M. S. Alvim, M. W. Hicks, and M. R. Clarkson. Quantifying information flow for dynamic secrets. In *SSP*, 2014.

[28] U. Maurer. Abstract models of computation in cryptography. In *Cryptography and Coding*, pages 1–12. Springer, 2005.

[29] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[30] P. J. Mucci, S. Browne, C. Deane, and G. Ho. Papi: A portable interface to hardware performance counters. In *DoD HPCMP Users Group Conference*, 1999.

[31] R. Shokri, G. Theodorakopoulos, C. Troncoso, J.-P. Hubaux, and J.-Y. Le Boudec. Protecting location privacy: optimal strategy against localization attacks. In *CCS*, pages 617–627. ACM, 2012.

[32] V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266. Springer, 1997.

[33] G. Smith. On the foundations of quantitative information flow. In *FoSSaCS*, pages 288–302. Springer, 2009.

[34] M. Tambe. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press, 2011.

[35] E. Zermelo. Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. In *Proc. Fifth International Congress of Mathematicians*, volume 2, pages 501–504. II, Cambridge UP, Cambridge, 1913.

[36] D. Zhang, A. Askarov, and A. C. Myers. Predictive mitigation of timing channels in interactive systems. In *CCS*, pages 563–574. ACM, 2011.