Securing Computation Against Leaky Hardware

Alexander Lam
15-453 Spring 2014

May 1, 2014

1 Introduction

Most algorithms are designed under the assumption that the computers which will run the algorithm are black-box secure. An intuitive notion of black-box security is that the computer will not leak anything about its internal state or computation to an outside observer. For example, most encryption algorithms are designed so that the encrypted output is secure so that no observer of the encrypted data can extract any meaningful information from it. However, these same algorithms assume that the data which is being encrypted and the encryption key will not be revealed to an observer when the encryption algorithm performs the encryption. If these assumptions are false, then the encryption algorithm provides no security at all.

Unfortunately, practical implementations of computers do reveal such information by "leaking" it. There are many possible "side-channel" attacks shown to work on actual computing devices. Observable side-effects of computing on digital computers include power usage, acoustic emissions, electro-magnetic interference, timing attacks, and performance measurements. [2, 3]

In order to defend against such attacks, there has been research into:

- Total obfuscation of the algorithm
- Algorithms designed with leaky hardware in mind
- Modifications to existing circuits to make them leak-proof
- Compilers for transforming an existing program into a leak-proof program [1]

2 Side Channel Attacks

There is a surprising amount of very practical research into exploiting leakage from modern digital computers. All the methods which I learned were able to develop a technique with which RSA private encryption keys could be extracted from a computer through repeated trials and statistical analysis in order to obtain enough information to extract the desired state.

Power Analysis The amount of electric current that a computer chip consumes at an instantaneous point in time is highly dependent on the type of operation it is performing and the data on which the operation is performed. An addition operation may have a different current signature from multiplication due to different circuitry. The data which is being operated on also changes the current signature - for example, flipping a flip-flop to the "ON" state may require more power than flipping it "OFF." By measuring the current consumption across time of a computer chip, it is possible to generate a statistical model which can reveal the chip's internal state. For example, this kind of analysis was carried out on a smart card, and the AES key of the smart card was extracted. [3]

It is possible to extend this sort of power analysis to measuring the ground potential of a computer. The more current a computer draws, the higher the ground potential is due to resistance in the ground path. This results in an indirect method of measuring the current draw of the computer that can be taken advantage of anywhere a ground path to the computer exists, which are more abundant than at first glance. Nearly every cable connected to a computer carries a ground conductor which can be utilized for this purpose. [2]

Acoustic Analysis Computer components vibrate during computing. This is mostly due to electro-mechanical phenomena such as varying magnetic fields in the inductors of the power supply system. By performing multiple trials recording audio of a computer performing a secure computation, it has been shown that statistical analysis can be applied to extract secret state - for example, an RSA key. This can be performed by placing a mobile phone next to the computer in question and using the phone's microphone [2]. Daniel Genkin's paper on this topic was featured on the popular website Slashdot.org, which is what prompted me to start learning about leaky computing.

L3 Cache Timing Another method shown to be able to extract an RSA key is by running another program on the computer and utilizing the timing information which leaks from performance variation due to processor cache [7]. This is a technique which requires no physical presence at the computer under observation.

3 Total Obfuscation

Total obfuscation of the algorithm or circuit which is to be run on leaky hardware is the ideal solution. Any adversary, no matter how powerful, given an unlimited amount of leaked information, would not be able to calculate any useful information from the leakage. However, it has been shown that total obfuscation is impossible in Boaz Barak's paper, " On the (Im)Possibility of Obfuscating Programs," which we summarize here. [4]

The model of total obfuscation is: An obfuscator $\mathcal{O}$ takes a Turing machine M and emits a new Turing machine $\mathcal{O}(M)$ which performs the same calculation as M but is obfuscated. By obfuscated, we mean that anything which can be computed efficiently from $\mathcal{O}(M)$ can also be computed efficiently using oracle access to M. Intuitively, any information that $\mathcal{O}(M)$ leaks is also leaked from strictly input-output (oracle) access to M, where no side-channel attacks can occur and no leakage can be measured.

Any leaked information which can be calculated from the execution of some Turing machine M can be denoted as a predicate π, where $\pi : M \to 0, 1$. Hence, π calculates one bit of information from some Turing machine M.

This notion of obfuscation is called the "Black Box" property. Formalized, this is: For every polynomial time probabalistic Turing Machine A which calculates π, there is a polynomial time probabalistic Turing Machine S with oracle access to M and a negligible function α such that for all Turing machines M,

$$\left|\Pr\left[A(\mathcal{O}(M)) = 1\right] - \Pr\left[S^M(1^{|M|}) = 1\right]\right| \leq \alpha(|M|)$$

Intuitively, $\Pr\left[A(\mathcal{O}(M) = 1\right]$ is the probability that the polytime probabalistic Turing machine A with access to the obfuscated version of M outputs 1, and $\Pr\left[S^M(1^{|M|}) = 1\right]$ is the probability that the polytime probabalistic Turing Machine S, with oracle access to M outputs 1. $\alpha(M)$ produces a probabalistically negligible value based on the length of M. Overall, the entire relation states that A can't calculate anything about $\mathcal{O}(M)$ with any better probability than S, which has oracle access to M instead of access to M directly.

$\mathcal{O}(M)$ should be polynomially larger with respect to M: For every Turing Machine M, $|\mathcal{O}(M)| < p(|M|)$, where p is some polynomial. $\mathcal{O}(M)$ should also be polynomially slower with respect to M: For every Turing Machine M, if $\mathcal{O}(M)$ takes s steps, then M should take at most $p(s)$ steps for some polynomial p.

In the paper, Boaz Barak et. al. use contradiction to show that such an obfuscator does not exist. This contradiction requires a 2-TM obfuscator, which works for two Turing Machines. This obfuscator is the same as the one presented above, except that the Black Box property is strengthed: For every polynomial time probabalistic Turing Machine A, there is a polynomial time probabalistic Turing Machine S with oracle access to M and N and a negligible function α such that for all Turing machines M, N,

$$\left|\Pr\left[A(\mathcal{O}(M), \mathcal{O}(N)) = 1\right] - \Pr\left[S^{M,N}(1^{|M|+|N|}) = 1\right]\right| \leq \alpha(|M|)$$

Say that there does exist such an obfuscator $\mathcal{O}$ which satisfies the above obfuscator properties. We define two Turing machines, where $\alpha, \beta \in \{0,1\}^k$:

$$C_{\alpha,\beta}(x) = \begin{cases} \beta & x = \alpha \\ 0^k & \text{otherwise} \end{cases} \quad D_{\alpha,\beta}(C) = \begin{cases} 1 & C(\alpha) = \beta \\ 0 & \text{otherwise} \end{cases}$$

We define our adversary polynomial time probabilistic Turing Machine $A(C, D)$ to be the machine which computes $D(C)$. Hence,

$$\Pr\left[A(\mathcal{O}(C_{\alpha,\beta}), \mathcal{O}(D_{\alpha,\beta})) = 1\right] = 1$$

because $\mathcal{O}(C_{\alpha,\beta})$ computes the same function as $C_{\alpha,\beta}$.

Any other polynomial time probabilistic Turing Machine S with oracle access to $C_{\alpha,\beta}$ and $D_{\alpha,\beta}$ has exponentially small probability (2^{-k}) of providing the correct input to the oracle of either function to cause non-zero output. So, if Z_k is the Turing machine which always outputs O^k, and α and β are randomly chosen, then for every S:

$$\left|\Pr\left[S^{C_{\alpha,\beta},D_{\alpha,\beta}}(1^k) = 1\right] - \Pr\left[S^{Z_k,D_{\alpha,\beta}}(1^k) = 1\right]\right| \leq 2^{-k}$$

However, we know that

$$\Pr\left[A(\mathcal{O}(Z_k), \mathcal{O}(D_{\alpha,\beta})) = 1\right] = 0$$

This is a contradiction. If we let

$$\left|\Pr\left[A(\mathcal{O}(C_{\alpha,\beta}), \mathcal{O}(D_{\alpha,\beta})) = 1\right] - \Pr\left[S^{C_{\alpha,\beta},D_{\alpha,\beta}}(1^k) = 1\right]\right| \leq \alpha(|M| + |N|)$$

be true, then it must be that $\Pr\left[S^{Z_k,D_{\alpha,\beta}}(1^k) = 1\right] \approx 1$. Then,

$$\left|\Pr\left[A(\mathcal{O}(Z_k, \mathcal{O}(D_{\alpha,\beta})) = 1\right] - \Pr\left[S^{Z_k,D_{\alpha,\beta}}(1^k) = 1\right]\right| \leq \alpha(|M| + |N|)$$

cannot be true because $\Pr\left[S^{Z_k,D_{\alpha,\beta}}(1^k) = 1\right] \approx 1$, and vica-versa.

Hence, no such obfuscator $\mathcal{O}$ exists. [4]

4 Describing A Pseudorandom Number Generator

Since total obfuscation is not possible, other approaches will have to be taken to produce secure computation. One of those is to directly design a algorithm, which is what Silvio Micali and Leonid Reyzin do in "Physically Observable Cryptography" [5]. The physical leakage model used here is a a relatively simplistic one - it assumes that all computation will leak its entire state, but that it is possible to create small hardware units which leak information according to some leakage function. These small hardware units, together with regular unsecured hardware, create a computer. The computer has a global memory which is accessible by all of the units, but each unit (a Turing Machine) also has its own local tape. This global memory does not leak information from unread addresses. If it did, then it would be impossible to hide anything as the entire contents of memory would leak even as the machine did nothing.

In order to understand the pseudo-random number generator described, we must introduce some terminology. First, define $\mathcal{P}$ to be a hardware unit which calculates the permutation f_p, x to be a random bit string of length k, and $\mathcal{P}(x) = y$. F is an adversary.

- A hardware unit $\mathcal{P}$ which calculates f_p, a one way function, is a "minimal one-way function" if for any polynomial-time adversary F, any state measured by $F(1^k)$ combined with y cannot be used by F to produce z such that $f_p(z) = y$. In other words, F cannot use any leakage of the output y to reverse the calculation of $\mathcal{P}$.
- A hardware unit $\mathcal{P}$ which calculates f_p, a one way function, is a "durable function" if it is a minimal one-way function such that the output of $\mathcal{P}$ is random even in the face of leakage. In other words, even armed with the state measured by $F(1^k)$, F cannot distinguish between y and some other random bit with any significant probability.
- A polynomial time hardware unit $\mathcal{P}$ is "unpredictable" if during its calculation, when i bits have been output, the leakage measured by the adversary F cannot be used to predict the $i+1$ bit of output with any significant probability.

Silvio Micali and Leonid Reyzin show that it is possible to create an unpredictable bit generator if there is a durable permutation function by mimicing the Blum-Micali construction. The random seed for the random number generator consists of x_0 and r. Using a durable one-way $\mathcal{P}$, compute the permuted intermediate bits $x_1 = \mathcal{P}(x_0)$, $x_2 = \mathcal{P}(x_1)$, ..., $x_n = \mathcal{P}(x_{n-1})$.

The Blum-Micali construction then calls for outputting the Goldreich-Levin bit of $x_n, ..., x_1$. The Goldreich-Levin bit calculation may leak all x_i, so to maintain unpredictability, the calculation of the Goldreich-Levin bit is done in reverse order, starting with $x_n \cdot r_n$ and ending with $x_0 \cdot r_0$. This reversal ensures that all computation using $\mathcal{P}$ is done before the Goldreich-Levin bit calculation begins. This is because when the Goldreich-Levin bit is calculated, r also leaks, but this is after all calculation of $\mathcal{P}$ is done, so it is too late to be of any use to the adversary, because the adversary needs r during the calculation of $\mathcal{P}$ to determine what to measure.

An overview of the proof by contradiction for one output bit of this construction: Let x be an input seed bit. Compute $\mathcal{P}(x) = y$, and then compute $r \cdot y$. The adversary has observed some state from the computation of $\mathcal{P}(x)$ and also has r. If the adversary can predict $r \cdot x$ with significant probability greater than $\frac{1}{2}$, then it can also predict $r \cdot x$ for any r with significant probability (because r is known during the prediction). This means that x can be deduced, contradicting the one-way minimal-ness of $\mathcal{P}$. We can extend this to the entire output of the construction because bits $r \cdot x_i$ are output in reverse order, so unpredictability is proven by this single-bit result.

5 Forming Secure Circuits

Other work focuses on modifying existing constructions to be secure against leakage. Yuval Ishai et. al. in "Private Circuits: Securing Hardware against Probing Attacks" [6] show how to modify any arbitrary boolean circuit to be secure against a special kind of adversary. The model of physical leakage is that the adversary has t wire probes which can probe any wire in the circuit. These wire probes can read the value of the wires being probed, and the adversary may move the wire probes in between time periods (clock cycles), but not during a single time period.

The paper presents a way to create a perfectly private circuit so that no t-strong adversary can extract any information. Any information that the adversary can extract can also be gleaned from a simulation of the circuit with no wire probes. In addition, the function which the modified circuit calculates should be indistinguishable from the original circuit.

Denote the original circuit to be C and the transformed circuit to be C'. Each wire in C will be represented in C' using $2t + 1$ wires. The new circuit will contain "random bit" gates, which do not have any input wires and have one output wire, carrying a random value. Before transforming C, C should be expressed using NOT gates and 2-input AND gates. It is possible to express all boolean circuits with these two primitives.

First, define an input encoder I. Each original input wire x is represented as $m+1$ wires, where $m = 2t$. m wires are random bit values $r_1, ...r_m$ produced by random bit gates, and the last wire is calculated as $r_{m+1} = x \oplus r_1 \oplus ... \oplus r_m$. Hence, taking $\bigoplus_{i=1}^{m+1} r_i = x$.

The output decoder O will produce an output wire which carries the same output value as in the original circuit. This is done by XORing together all of the $m+1$ wires representing the original wire: $y = \bigoplus_{i=1}^{m+1} r_i$.

Since the adversary only has t wire probes, it cannot possibly obtain the value that $m+1$ wires represent in the original circuit, because each wire is independently and uniformly distributed - only with all $m+1$ wires does any useful information appear. (It is possible to secure the circuit using only $t+1$ wires to represent a wire in the original circuit, but it is harder to understand).

The circuit transformer T transforms NOT and 2-input AND gates in C into secure versions in C'. Transforming a NOT gate is trivial - place a one-input one-output NOT gate on one of the $m+1$ wires in C' that representing a single wire in C. Since XOR is addition of the bits modulo 2, flipping a single wire also flips the value of $\bigoplus_i w_i = \Sigma_i \ w_i \mod 2$, where w_i is the i-th wire.

An AND gate has inputs a and b and output c. In C', the wires representing these inputs and outputs are $a_1, ...a_{m+1}$, $c_1, ...c_{m+1}$, and $c_1, ...c_{m+1}$. Note that $a = \Sigma_i \ a_i \mod 2$ and $b = \Sigma_i \ b_i \mod 2$, so $c = \Sigma_{i,j} \ a_i b_j \mod 2$. Since the AND gate will be implemented using multiple gates and internal wires, a scheme is needed to ensure that the internal wires do not leak any information.

The internal wires carry intermediate values $z_{i,j}$ for $i \neq j$. For each i, j such that $1 \leq i < j \leq m+1$, $z_{i,j}$ is a value generated by a random-bit gate, and $z_{j,i} = (z_{i,j} \oplus a_i b_j) \oplus a_j b_i$. This allows us to keep each $z_{i,j}$ and $z_{j,i}$ random, so these wires will not leak.

The output bits of the AND gate are calculated as

$$c_i = a_i b_i \oplus \bigoplus_{j \neq i} z_{i,j}$$

Note that each c_i is now also random, because no c_i will be composed of both $z_{i,j}$ and $z_{j,i}$ for $i \neq j$. If all c_i are XORed together as $c = \bigoplus_i c_i$, all the random bits $z_{k,l}$ for $k < l$ cancel out, leaving $c = \bigoplus_{i,j} a_i b_j$, which is our desired result.

Any circuit C can be transformed into a t-secure circuit $C' = T(O(I(C)))$ with this construction. The transformed AND gate expands to $O(m^2)$ gates, so if every gate in C is an AND gate, and there are n gates, the gate cost of this transformation is $O(nm^2)$.

6 Other Work

There are more advanced techniques for securing computation against leakage, but most of these are out of technical reach for me. For example, Guy N Rothblum in "How to Compute under AC^0 Leakage without Secure Hardware" develops a compiler to prevent leakage of a computational complexity class AC^0 [1]. However, I lack the knowledge required to understand the implications of computational complexity classes. I also have not had significant cryptographic experience prior to this project, so I spent a lot of time learning about cryptography before being able to understand the concepts presented in the papers I read.

I have not seen a paper which attempts to apply any of the techniques set forth for preventing leakage to an actual side-channel attack, which leads me to suspect that there may be a lack of research using a model which accurately describes the leakage seen in a modern digital computer.

References

[1] Guy N Rothblum: How to Compute under AC^0 Leakage without Secure Hardware. In: CRYPTO, vol 7417, pp. 552-569. Springer, 2012

[2] Daniel Genkin, Adi Shamir, Eran Tromer: RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. In: Cryptology ePrint Archive, Report 2013/857

[3] Paul Kocher, Joshua Jaffe, Benjamin Jun: Introduction to differential power analysis. In: Journal of Cryptographic Engineering, vol 1, pp. 5-27. Springer-Verlag, 2011

[4] Boaz Barak, Oded Goldreich, Russel Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, Ke Yang: On the (Im)Possibility of Obfuscating Programs. In: J. ACM, vol 59, no 2, pp.6:1-6:48. ACM, 2012

[5] Silvio Micali, Leonid Reyzin: Physically Observable Cryptography. In: Theory of Cryptography, vol 2951, pp. 278-296. Springer Berlin Heidelberg, 2004

[6] Yuval Ishai, Amit Sahai, David Wagner: Private Circuits: Securing Hardware against Probing Attacks. In: Advances in Cryptology - CRYPTO 2003, vol 2729, pp. 463-481. Springer Berlin Heidelberg, 2003

[7] Yuval Yarom, Katrina E. Falkner: Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack. In: IACR Cryptology ePrint Archive, vol 2013, pp. 448. 2013