Editors: Peter Gutmann, pgut001@cs.auckland.ac.nz
David Naccache, david.naccache@ens.fr
Charles C. Palmer, ccpalmer@us.ibm.com

Micro-Architectural Cryptanalysis

Onur Aciiçmez and Jean-Pierre Seifert
Samsung Electronics R&D Center

Çetin Kaya Koç
Oregon State University

Cryptanalysis is the study of the methods used to obtain the meaning of encrypted information in a cryptosystem (typically, by finding a secret key)—in nontechnical terms, it's also called "code breaking" or "cracking the code." However, conventional cryptanalysis methods examine only cryptosystems' *theoretical* foundations—calculating how much effort is needed to factorize a given 1,024-bit RSA modulus, for example, falls under this category. But even though such questions are important, they never accurately reflect the strength of real systems in practice. In reality, an adversary can observe several different channels of information in addition to a cryptosystem's inputs, outputs, and public parameters. In other words, a flesh-and-blood adversary can gather much more information than the famous theoretical adversary, Eve.

Moreover, none of us are as lucky as Alice or Bob, who have the luxury of using the theoretical Turing machines described in textbooks. Instead, we use real devices, such as computers, smart cards, and cell phones, all of which reveal valuable information to adversaries, simply by letting them observe physical characteristics such as execution time,[1] power consumption,[2] and electromagnetic dissipation.[3] We call these information channels *side channels*, and they're clearly not among the input and output channels that Eve is used to observing.

This installment of Crypto Corner looks at some specific new side channels that are enabled by the very sophisticated ingredients of modern microprocessor architectures.

Side-channel cryptanalysis

Early side-channel studies initially focused mainly on smart cards. Compared to general computer systems, smart-card architectures are quite simple and thus an easy target for side-channel attacks. The security community thought these attacks couldn't be applied to general computer systems, but a timing attack on a Web server in 2003 changed this perspective.[4] Researchers showed that such an attack could even compromise remote systems over a network, which is very different from applying side-channel attacks on smart cards that are physically in your possession. Improvements on the original remote timing attack made it even more practical.[5]

Recent studies on side-channel analysis have led to a new area, *micro-architectural analysis*, which studies the effects of common processor components on cryptosystem security. Researchers realized that microprocessor component functionalities generate easily observable, data-dependent effects—a crypto algorithm's execution, for example, leaves "footprints" on the persistent state of data caches, instruction caches, and branch prediction units. These easy-to-see footprints depend on the operations performed during execution as well the data used in them, so an adversary could break a cryptosystem simply by running in parallel a so-called *spy process* to trace the footprints during or after the algorithm's execution. It's important to note that spy processes run in full isolation and can't directly read any data from the cryptosystem, but nevertheless, leaked footprints can lead to dramatic security breaches.

We've seen many examples of such micro-architectural attacks on widely used cryptosystems such as AES[6,7] and RSA.[8,9] Initial studies on cache attacks weren't really oriented around spy processes, but the rise of spy-oriented cache attacks in 2005 resulted in a whole new paradigm. Let's look more closely at how these attacks work.

The birth of micro-architectural analysis

A cache is a small storage area that a CPU uses to reduce average memory access time. By acting as a buffer between the main memory and the processor core, it provides the processor with fast and easy access to the most frequently used data (including instructions) without frequent external bus accesses. When the processor needs to read a location in main memory, it first checks to see if the data is already in the cache. If the data's there (a cache *hit*), the processor immediately uses it rather than accessing the main memory, which has a significantly longer latency; otherwise, it reads the data from the memory and stores a copy in the cache (a cache *miss*). When the AES process encrypts or decrypts a block of data, for example, it writes those parts of the lookup tables accessed during execution into cache.

However, this functionality also enables adversaries to launch different kinds of cache-based attacks on cryptosystems. Let's look at a simplified example on AES. An adversary runs a spy process that reads a large amount of data from private memory (from a large array, for instance). As a result of this read operation, the cache contains only some data from the spy's memory, if the array is big enough; from this point forward, the spy process releases the CPU, which might continue or start an AES operation. The CPU writes those parts of the lookup tables touched during the encryption into the cache, which overwrites some of the spy's data. Touched parts of the AES lookup tables stay in the cache even after the AES process finishes its execution—in other words, the cache contains the AES execution's footprints after its termination. If the adversary runs the spy process shortly after the AES terminates, the spy can trace these footprints—it simply reads its own private memory, but this time, it also measures the time it takes to read each small portion of the array. Because it takes more time to read the portions overwritten during AES execution, the spy process can determine which parts of the cache the AES modified. This information also reveals which parts of the lookup tables were accessed during encryption. If the adversary can gather this kind of information from several executions of AES, it's extremely easy to find the secret key used in the cipher.

In this example, we don't have to assume that these processes run on a simultaneous multithreading (SMT) processor; SMT architectures execute multiple processes simultaneously by special hardware support, thus on SMT architectures, the adversary can observe cipher execution "on the fly." Here, the spy process runs simultaneously with the cipher process (as opposed to running after the cipher process terminates), so an adversary can analyze, for example, each individual AES round and determine which parts of the cache were modified in each round instead of during the entire AES execution.

Clearly, this approach provides finer-grained information to the adversary and makes it much easier to break the cipher. Information from fewer than 15 executions is enough for SMT-based attacks to break 128-bit AES.[7] Furthermore, the recent introduction of a new attack type that seems impossible to launch on non-SMT microprocessors has demonstrated the severity of the security weaknesses in SMT functionalities.[10]

Unleashing the power of spies

Pure software-based side-channel attacks, including data cache attacks, rely on statistical methods and require many computational measurements under the same key. In late 2006, a new and very powerful attack type—simple branch prediction—attracted significant attention immediately after its introduction.[9] This spy-oriented attack has the power to extract almost all the key bits from *a single* RSA execution by accurately revealing the cryptosystem's execution path. A recent manuscript introduced another related micro-architectural attack type—instruction cache analysis—that also reveals a process's execution path by simply observing a single execution.[8] The successful extraction of enough information from a single execution rendered many already deployed side-channel mitigations totally useless.

Branch prediction attacks[9,11] and instruction cache attacks[8] differ from basic data cache attacks, but they rely on many of the same principles. These attacks try to determine a cryptosystem's execution path instead of data access patterns, which are the focus of data cache attacks. The execution paths of public-key cryptosystems such as RSA usually depend on the key's secret parameters. Therefore, once adversaries obtain the execution path, they can reconstruct the private key from this information. Another novel application of execution path observation attacks is its application to the binary extended Euclidean algorithm (BEEA), which is frequently used for RSA key setup and certain real-world RSA side-channel protections.[12]

The instruction cache is a dedicated cache inside a CPU that stores the instructions of recently executed code sections. In instruction cache attacks, the spy process executes dummy instructions to overwrite the instruction cache, which is fundamentally different from reading a large array to overwrite the data cache, which is what happens in data cache attacks. Similar to the basic cache attack method we described in the previous section, an adversary can run such a spy process to execute dummy instructions in parallel with a cipher execution and thereby trace the cipher's footprints. In instruction cache attacks, these footprints reveal which instructions were executed inside the cipher process—in other words, they also reveal the cipher's execution path.

The branch prediction mechanism is a very crucial part of all general-purpose microprocessors. To complete at least one instruction per clock cycle, those highly pipelined microprocessors must guess the most probable execution path before the actual and final retirement of a conditional branch and thus speculatively feed the pipeline with the next instruction. However, if the speculatively executed instruction flow turns out to be wrong, the execution suffers from a *misprediction* delay because the CPU must flush the entire pipeline and resume execution from the correct instruction after the current branch instruction.

Of the different types of branch prediction attacks, the most powerful is the simple branch prediction attack (SBPA).[9] SBPA targets the branch target buffer (BTB), which is inside a branch prediction unit (BPU). The BTB stores data related to recently executed branches; similar to a cache miss, if the CPU can't find the corresponding data inside the BTB, the branch's execution takes longer. An adversary can exploit this fact by running a spy process that observes the changes inside the BTB during the cryptosystem's execution. To achieve this functionality, the spy process must execute specialized sequences of several conditional branches and measure the overall time.

Micro-architectural analysis of cryptosystem implementations is a promising and interesting new security research direction that will continue to grow. Because simple branch prediction attacks and instruction cache attacks can extract almost all the RSA key bits by observing a single RSA operation, they're quite dangerous for real-world crypto implementations, such as OpenSSL, which runs on more than 60 percent of the world's server installations. However, the actual threat of micro-architectural analysis isn't limited just to cryptographic algorithms: all the pure software attacks mentioned here allow a totally unprivileged process to compromise the security of microprocessor-based computing environments.

The recent efforts in building trusted computing platforms increase the significance of these threats. Such attacks can function even in the presence of sophisticated security mechanisms such as memory protection, sandboxing, or even virtualization because they exploit deep processor ingredients below the trust architecture boundary. It's therefore necessary to carefully consider every piece of a platform, from the highest level all the way down to the deepest ingredients, to achieve the goal of really secure and trusted platforms and to avoid building a fortress on sand.

Unfortunately, trusted computing efforts address only architectural boundaries and provide no micro-architectural confinement of critical information. Initiatives that enable a simple CPU integration of crypto accelerators, such as AMD's Torrenza and TPM chips, are definitely a promising and effective first step to increase the strengths of the systems against micro-architectural attacks. However, we have to do a lot more to completely overcome these threats. In particular, reconsidering the micro-architectural designs would be a very positive approach. ☐

References

1. P.C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," *Advances in Cryptology* (CRYPTO 96), LNCS 1109, N. Koblitz, ed., Springer-Verlag, 1996, pp. 104–113.
2. P.C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Advances in Cryptology* (CRYPTO 99), LNCS 1666, M. Wiener, ed., Springer-Verlag, 1999, pp. 388–397.
3. K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic Analysis: Concrete Results," *Cryptographic Hardware and Embedded Systems* (CHES 01), LNCS 2162, Ç.K. Koç, D. Naccache, and C. Paar, eds., Springer-Verlag, 2001, pp. 251–261.
4. D. Brumley and D. Boneh, "Remote Timing Attacks are Practical," *Proc. 12th Usenix Security Symp.*, Usenix Assoc., 2003, pp. 1–14.
5. O. Acıiçmez, W. Schindler, and Ç.K. Koç, "Improving Brumley and Boneh Timing Attack on Unprotected SSL Implementations," *Proc. 12th ACM Conf. Computer and Comm. Security* (ACM-CCS 05), C. Meadows and P. Syverson, eds., ACM Press, 2005, pp. 139–146.
6. O. Acıiçmez, W. Schindler, and Ç.K. Koç, "Cache-Based Remote Timing Attack on the AES," *Cryptographers' Track at the RSA Conf.* (CT-RSA 07), LNCS 4377, M. Abe, ed., Springer-Verlag, 2007, pp. 271–286.
7. M. Neve and J.-P. Seifert, "Advances on Access-Driven Cache Attacks on AES," to be published in *Selected Areas of Cryptography* (SAC 06), 2007.
8. O. Acıiçmez, "Yet Another Micro-Architectural Attack: Exploiting I-Cache," *Cryptology ePrint Archive*, report 2007/164, May 2007.
9. O. Acıiçmez, Ç.K. Koç, and J.-P. Seifert, "On the Power of Simple Branch Prediction Analysis," to be published in *ACM Symp. Inform-Ation, Computer and Comm. Security* (ASIACCS 07), 2007.
10. O. Acıiçmez and J.-P. Seifert, "Cheap Hardware Parallelism Implies Cheap Security," to be published in *Fault Diagnosis and Tolerance in Cryptography* (FDCT 07), IEEE CS Press, 2007.
11. O. Acıiçmez, Ç.K. Koç, and J.-P. Seifert, "Predicting Secret Keys via Branch Prediction," *Cryptographers' Track at the RSA Conf.* (CT-RSA 07), LNCS 4377, M. Abe, ed., Springer-Verlag, 2007, pp. 225–242.
12. O. Acıiçmez, S. Gueron, and J.-P. Seifert, "New Branch Prediction Vulnerabilities in OpenSSL and Necessary Software Countermeasures," *Cryptology ePrint Archive*, report 2007/039, Feb. 2007.

***Onur Acıiçmez** is a research scientist at Samsung.His technical interests include trusted computing, side-channel analysis, implementation aspects of security technologies, and computer and information security in general. Acıiçmez has a PhD in electrical engineering and computer science from Oregon State University. Contact him at onur.aciicmez@gmail.com.*

***Jean-Pierre Seifert** is a principal engineer at Samsung. His technical interests include all aspects of computer and information security in general. Seifert has a PhD in mathematics from the Johan-Wolfgang Goethe University and is a BIT-professor at Bolzano, Innsbruck, and Trento. Contact him at jeanpierre seifert@yahoo.com.*

***Çetin Kaya Koç** is a professor at Oregon State University. His technical interests include computer architectures and embedded systems, cryptographic engineering, and information security. Koç has a PhD in electrical and computer engineering from the University of California, Santa Barbara. He is a fellow of the IEEE. Contact him at koc@cryptocode.net.*