Hakin9 EXTRA

Issue 6/2012 (13) ISSN 1733-7186

TIMING ATTACKS IN AES

SIDE CHANNEL ATTACKS

CACHE TIMING ATTACKS

AUTOMATED ALGEBRAIC CRYPTANALYSIS

TIMING ATTACK AGAINST THE CBC OPERATING MODE

PLUS

TIMING ATTACKS AGAINST RSA ENCRYPTION AND DECRYPTION REVISITED

Atola Insight

That's all you need for data recovery.

Atola Technology offers *Atola Insight* – the only data recovery device that covers the entire data recovery process: *in-depth* **HDD diagnostics**, **firmware recovery**, **HDD duplication**, and **file recovery**. It is like a whole data recovery Lab in one Tool.

This product is the best choice for seasoned professionals as well as start-up data recovery companies.

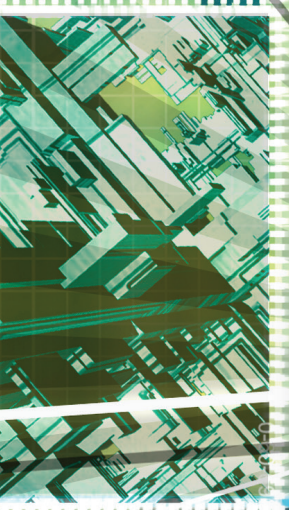
Emphasized features at a glance:

- Automatic in-depth diagnostic of all hard drive components
- Automatic firmware recovery and ATA password removal
- Very fast imaging of damaged drives
- Imaging by heads
- Case management
- Real time current monitor
- Firmware area backup system
- Serial port and power control
- Write protection switch

Atola TECHNOLOGY

Visit atola.com for details

Air Freshener?



Printer PSU?
...nope

The Industry's First Commercial Pentesting Drop Box

THE **PWN Plug**

FEATURES:

- Covert tunneling
- SSH access over 3G/GSM cell networks
- NAC/802.1x bypass
- and more!

Discover the glory of
Universal Plug & Pwn

@ pwnieexpress.com

PWNIE EXPRESS

t) @pwnieexpress **e)** info@pwnieexpress.com **p)** 802.227.2PWN

HAKIN9 EXTRA

Managing:
Michał Wiśniewski
m.wisniewski@software.com.pl

Senior Consultant/Publisher:
Paweł Marciniak

Editor in Chief:
Grzegorz Tabaka
grzegorz.tabaka@hakin9.org

Art Director:
Marcin Ziółkowski

DTP:
Marcin Ziółkowski
www.gdstudio.pl

Production Director:
Andrzej Kuca
andrzej.kuca@hakin9.org

Marketing Director:
Grzegorz Tabaka
grzegorz.tabaka@hakin9.org

Proofreadres:
Dan Dieterle, Michael Munt,
Michał Wiśniewski

Top Betatesters:
Ruggero Rissone,
David von Vistauxx,
Dan Dieterle,
Johnette Moody,
Nick Baronian,
Dan Walsh,
Sanjay Bhalerao,
Jonathan Ringler,
Arnoud Tijssen,
Patrik Gange

Publisher: Hakin9 Media Sp. z o.o. SK
02-682 Warszawa, ul. Bokserska 1
www.hakin9.org/en

Dear Readers,

This month we decided to prepare a spacious issue on Timing Attacks. There are two reasons for that: first – as an "extra" branch of hakin9 we search for the hottest topics in IT-Security and we enjoy expanding on the topics that we have prepared. The second reason, however, has everything to do with the launch of the newly established Cryptomag. We are preparing completely new magazine, independent of hakin9, and solely devoted to cryptography (as its name suggests). Stay tuned to hakin9 news and be ready for the new magazine when it appears. Below is what we have prepared for you in this month's hakin9 Extra. Vincent Rijmen in his article on "Timing Attacks on AES" will show you how the execution time of an AES encryption can be used to derive the secret key. Qi Chai, in this issue's special article, will re-visit Timing Attacks against RSA. Weizhong Yang and Jeffrey Zheng are going to present Variant Pseudo-Random Number Generator. Michael W. Farb, Yue-Hsun Lin, Adrian Perrig and Jonathan McCune are going to expatiate on Safeslinger – an easy-to-use and secure public-key exchange. Martin Rublik, or regular collaborator is going to present an Overview of Side Channel and Timing Attacks. In an article entitled "The Dichotomy of Symmetric vs Asymmetric Cryptography" Wayne Patterson discusses the fundamental dilemma of the two kinds of cryptography in today's use. Matthieu Bontrond is going to present Timing Attack against CBC operating mode – an attack that enables decryption of blocks without attacking the encryption key. Theodosis Mourouzis has presented us Automated Algebraic Cryptanalysis. Michael Wisher presented his expertise on "Cache-Timing Attacks on Symmetric Cryptographic Primitives". Nitin Jain is going to present you the article on "Timing Attacks on Practical Quantum Cryptographic Systems. The last, but not least is the interview with Vitaliy Mokosiy – Atola's Bandura Project Manager and key developer.

I hope that you will enjoy the reading!

Michał Wiśniewski, hakin9 Extra
m.wisniewski@software.com.pl

MONITOR STRONY

Innowacyjne e-usługi do monitorowania stron www

SEOmonitor
monitorowanie strony www na potrzeby SEO

SPEEDmonitor
monitorowanie prędkości ładowania strony www

CONTENTmonitor
monitorowanie poprawności językowe treści publikowanych na stronie www

www.monitorstrony.pl

INNOWACYJNA GOSPODARKA
NARODOWA STRATEGIA SPÓJNOŚCI

DOTACJE NA INNOWACJE
Projekt współfinansowany przez Unię Europejską
z Europejskiego Funduszu Rozwoju Regionalnego

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ
ROZWOJU REGIONALNEGO

TIMING ATTACKS ON AES

VINCENT RIJMEN

Abstract: The black-box security of AES remains unchallenged. Nobody is able to decrypt ciphertexts that have been encrypted with AES and neither is there a method known to recover the secret key, even when a large amount of messages and the corresponding ciphertexts are known. However, if the attacker has access to additional information about the internal operations of AES, then practical attacks are sometimes possible. In this article, we explain how the execution time of an AES encryption can be used to derive the secret key.

Introduction

It has been known since a long time that programs can pass on hidden information deliberately by varying their consumption of CPU resources, e.g. execution time or RAM usage. However, only in 1996 Paul Kocher published the first article on *timing attacks*: methods that recover the secret key of otherwise practically unbreakable cryptographic algorithms by exploiting detailed information on their execution time [4].

In this article, we explain two timing attacks on AES. Firstly, by way of introduction, we show how a naive implementation of the finite field operations used in the MixColumns step of AES leads to a simple attack. This attack can also be avoided easily. Next, we show an attack based on the timing differences caused by the working of cache memory. The attack assumes that an attacker can make accurate timing measurements and requires a bit more analysis, but is also more difficult to counter.

The execution time of a program is just one type of *side-channel information*. In particular for implementations in hardware and on very simple processors (think: smartcards), researchers discovered powerful attacks based on measurements of the power consumption or the electro-magnetic radiation during an encryption operation. Since those attacks require a thorough understanding of the design of electronic circuits, we won't cover them here.

We assume that the reader is familiar with the AES (Advanced Encryption Standard). Otherwise, good descriptions can be found in the FIPS standard [1] and in the AES book [2].

Simple case: xtime

An implementation of MixColumns

The MixColumns step of the AES round transformation contains multiplication operations in the finite field GF(256). Each byte of the input is multiplied by the constants **1**, $\boldsymbol{x}$ and $\boldsymbol{x} \oplus \mathbf{1}$. (The last two constants are often denoted by **2** and **3**.) On a typical processor, these multiplications are implemented by means of a table lookup. In this example, however, we assume that the implementation explicitly computes the multiplications. This could be the case, for instance, if the processor has so little RAM (or cache) available, that we don't want to store this table.

The multiplication by $\boldsymbol{x}$ can be implemented as shown in Algorithm 1. Note that multiplication by $\boldsymbol{x} \oplus \mathbf{1}$ can be implemented by using the law of distributivity in GF(256): $a \times (\boldsymbol{x} \oplus \mathbf{1}) = (a \times \boldsymbol{x}) \oplus (a \times \mathbf{1}) = (a \times \boldsymbol{x}) \oplus a$. Hence, an implementation of xtime and some xor operations are all that we need to implement MixColumns.

Visual inspection of Algorithm 1 reveals that on a simple processor without advanced scheduling tricks, there will be a noticeable variation in the execution time of this routine. If the MSB of a is set, then the routine will take longer, because that branch contains extra instructions.

A timing attack consists now essentially of three phases:

- Trigger AES encryptions of different message blocks. Measure and record the execution times.

- Derive from these measurements information on the MSBs of the input bytes of a MixColumns step.
- Convert this information into knowledge about the secret key.

The timing attack in detail

We will now go in detail through each of the phases of the attack. The first phase assumes that we can obtain timings of individual AES encryption operations, i.e. we need to be able to time the encryption of single 128-bit blocks. In some cases, this is easy. For example, if we want to extract a key from a smartcard that we own, a relatively cheap oscilloscope will allow to time individual encryption operations. In other cases, it will be impossible to avoid noise on the measurements. A certain level of noise can be eliminated by averaging different measurements. If we can't obtain enough timings with a sufficient accuracy, then the attack will work with only a low probability.

For the second phase, assume that we recorded the execution times of several hundreds of encryptions. Let $P[0]$, $P[1]$, …, $P[15]$ denote the 16 bytes of one message block. For the attack, it is needed that the attacker knows at least one byte at a fixed position for all the encryptions that he measured. We assume here that this is byte $P[0]$. Let $K[0]$, $K[1]$, …, $K[15]$ denote the first 16 bytes of the key and let SB() denote the substitution operation (S-box) used in the SubBytes step of AES.

We consider now the MixColumns step in the first round of an AES encryption. The input bytes of this step are given by:

$$Q[i] = \mathrm{SB}(P[i] \oplus K[i]), \text{ for } i = 0, 1, \ldots, 15.$$

We group the plaintexts according to their value of $P[0]$ and compute the average execution time in each group. Since $K[0]$ is a constant, albeit an unknown constant, we know from (1) that within each group the value of $Q[0]$ is constant.

If the multiplication by $\boldsymbol{x}$ is implemented like in Algorithm 1, then the multiplication of $Q[0]$ by $\boldsymbol{x}$ will take more time when the MSB of $Q[0]$ is set. Of course, during each AES encryption there are 9, 11 or 13 applications of MixColumns (for 10, 12 or 14 rounds). Each MixColumns contains 16 multiplications by $\boldsymbol{x}$, which all have their execution time depend of the MSB of their input. Hence, there is no deterministic relation between the execution time of an encryption and the MSB of $Q[0]$. However, there will be a statistical correlation. By measuring enough execution times, the effects of all the multiplications will cancel out on average, except for the execution time of the multiplication of $Q[0]$, because we keep that value constant within each group. In theory, we can even average out the noise on our measurements coming from the processor, the network, …

Let timing[i] denote the average execution time for the plaintexts with $P[0]=i$. If all went well, 128 values in timing[i] should be significantly higher than the 128 remaining values, corresponding to the 128 values of $Q[0]$ with the MSB set, respectively the 128 values of $Q[0]$ with the MSB cleared. We will then run Algorithm 2 to determine the most likely value of $K[0]$. We can reuse the timing information to determine the key bytes $K[1]$, $K[2]$, …, $K[15]$, provided that we know the plaintext bytes $P[1]$, $P[2]$, …, $P[15]$.

Extensions

If we don't know all 16 plaintext bytes, or if the key is longer than 128 bits, then there still remain bytes of the key to be determined. Those bytes can be extracted by analyzing the MixColumns of the second round of AES. Every input byte of this MixColumns operation depends on 4 plaintext bytes and 5 bytes of the key, of which we hopefully already recovered some. An analysis similar to the one we just did, but slightly more involved, will allow to determine the remaining bytes of the key. We refer the interested reader to [5].

In many applications of encryption, the attacker doesn't have access to the unencrypted messages. However, usually the attacker can access the ciphertexts. (Why else would the application designer want to use encryption?) The timing attack can be adapted to work with ciphertexts instead of plaintexts. In that case, we will sort the ciphertexts according to the value of a ciphertext byte and try to trigger decryptions instead of encryptions. We target now the InvMixColumns step closest to the ciphertext and try to recover the round key that is xor-ed to the ciphertext at the start of the decryption. In the InvMixColumns step, each byte of the input is multiplied by the constants $\boldsymbol{x}^3 \oplus \mathbf{1}$, $\boldsymbol{x}^3 \oplus \boldsymbol{x} \oplus \mathbf{1}$, $\boldsymbol{x}^3 \oplus \boldsymbol{x}^2 \oplus \mathbf{1}$ and $\boldsymbol{x}^3 \oplus \boldsymbol{x}^2 \oplus \boldsymbol{x}$. (Often denoted by **9**, **B**, **D** and **E**.) There are several ways to implement InvMixColumns, each with their own relation between input values and execution time. Without detailed knowledge on the implementation, we can't fully characterize the execution time distribution. However, in all sensible implementations that use xtime, the `small' values $\mathbf{0}$, $\mathbf{1}$, $\boldsymbol{x}$, …, $\boldsymbol{x}^3 \oplus \boldsymbol{x}^2 \oplus \boldsymbol{x} \oplus \mathbf{1}$ lead to the shortest execution times. Hence, we can run the attack successfully, perhaps using a little more AES operations than in the case where we have access to the unencrypted message.

Advanced case: cache timing attacks

In order to avoid the attack described in the previous section, the implementer can try to make sure that all conditional branches execute in the same time.

Provided that the processor allows to specify exactly which instructions have to be scheduled and executed in what order (deterministically), this task is not too difficult. There are however more possible caused why a program may exhibit a variable execution time. For example, a modern not-too-simple processor will have some cache memory.

Effects of the cache

The execution time of some instructions, for instance, a table lookup, depends on the state of the cache: if the requested value of the table is not in the cache, then it first has to be loaded into cache before it can be passed on to the processor. The relative time difference between a table lookup with cache hit and a table lookup with cache miss, can be quite large.

We will now describe how such a cache timing attack may work. There exist several different attack methods based on detecting cache misses. Our description here is based on the *Prime+Probe* attack by Tromer, Osvik and Shamir [5]. Modern caches use quite advanced rules to decide how long data and instructions are kept in the cache. We will start our description of the attack using a simplified cache architecture. Subsequently, we will adapt the attack for the cache architectures currently in use.

In the simplest model of a cache, each address in the RAM corresponds to one address in the cache. If the size of the cache is C, then data which resides at address i in the RAM, will be loaded at address $i \bmod C$ in the cache. If subsequently data from address $i + C$ is loaded into the cache, then it will evict the previously loaded data at address $i \bmod C$. Hence, if we create an array A with size C, and we execute code which

reads all elements of this array, then the cache will be *flushed*: the data of the array has kicked out all data and instructions previously present in the cache.

In a bird's eye view, a cache timing attack proceeds along similar lines as the timing attack explained in the previous section. Firstly, we measure carefully the execution time of an AES operation. Secondly, we derive from those measurements information on the input bytes of some SubBytes step. Finally, we convert that information into knowledge about the secret key.

The cache attack in detail

Assume that we have the privileges necessary to run programs on a processor where another user or process is performing AES operations using a key that we want to recover. Assume that we can trigger AES encryptions of blocks where we know at least one byte, e.g. byte $P[0]$. Assume that the SubBytes substitution SB() has been implemented using lookups in an array of 256 bytes. We denote this array by S. Further, assume that the array S is stored in contiguous memory and that we know the starting address of S. (If we don't know this starting address, then we will make a guess and try to run the attack. If the attack fails, then we have to guess again.)

We write a program that does only two things. Firstly, it creates the array A which has the same size as the cache, which is stored in contiguous memory, and which has the same starting address as S. Secondly, the program reads all values of A. We call this program Prime. In the measurement phase of the timing attack we will repeatedly perform the following three actions:

- Run the program Prime.
- Trigger the encryption of one block.
- Choose an arbitrary y in {0, 1, 2, ... ,255}. Read $A[y]$ and record the time it takes to do this.

It is important that the three actions follow each other closely. Ideally, they run as one atomic operation, but in reality we usually can't ensure this. If the sequence of actions is interrupted by other programs, then this will add noise to our measurements.

In our simplified cache architecture, the program Prime cleans all AES-related data from the cache. The second step will load again some AES-related data into the cache. The attack works only if this second step doesn't always load the full array S into cache. In the third step, we will try to determine whether array entry $S[y]$ was loaded into the cache during the second step. The reasoning is that if the operation to read $A[y]$ completes fast, then this must mean that $A[y]$ was still in cache. If $A[y]$ was not overwritten by the value $S[y]$, then this must be because the encryption operation in the second step didn't compute SB(y).

After repeating the steps 1-2-3 a few hundreds of times, we compute for k = 0, 1, ..., 255 the average reading time of $A[k$ Å $P[0]]$. Since the SubBytes in the first round applies SB to the value $K[0]$ Å $P[0]$, an AES encryption will always evict the entry $A[K[0]$ Å $P[0]]$ from the cache. Hence, the value $k = K[0]$ will be the k-value with the largest average reading time.

Cache lines

If we want to keep the number of encryptions as small as possible – which we usually do –, then we can repeat the third step for several y values. We have to make sure that we don't choose the same y twice for the same encryption, because after $A[y]$ has been read the first time, it will definitely be in the cache, and the second time measurement will not produce useful data. We now slightly modernize our model of the cache by realizing that current cache architectures don't load the data from RAM byte by byte. Instead, a whole *line* is loaded into cache. Hence, if multiple entries $S[y]$ are in the same cache line, then after one $A[y]$ has been read, we can't obtain useful data on any of the $S[y]$ in the same cache line until we trigger a new encryption. In fact, the situation is worse: since the attack can't distinguish between different $S[y]$ values on the same cache line, we won't be able to determine the key uniquely, no matter how many encryptions we trigger. If we assume that the starting address of S coincides with the start of a cache line (a compiler will always try to make this true), then the number of $S[y]$ values that share a cache line, will always be the same. Denoting this number by d, we have to admit that the attack isn't able to recover the lower $\log_2 d$ bits of the key.

A typical cache has cache lines of 64 bytes. Hence, one could store 64 entries $S[y]$ on the same cache line. Luckily, most AES implementations use the so-called T-tables, which implement a combination of the steps SubBytes and MixColumns. The T-tables use 4 bytes per $S[y]$, hence for these applications d is reduced to 64/4 = 16. This means that we can recover only the 8 - $\log_2$ 16 = 4 most significant bits of $K[0]$. Even if we know all 16 bytes of the plaintext, we can recover only 16 × 4 = 64 bits of the key. However, in the same way as with the simple timing attack described in the previous section, it is possible to target the SubBytes step in the second round of the AES encryption in order to recover more bits of the key. Also similar as in the previous section, we won't cover that extension in this article, but refer the interested reader to [5].

Associative caches

For the final update of our cache model and the attack, we need to consider associativity of caches. *W-way set-associative caches* are divided into sets, each containing W lines. Data which resides at address i in the RAM, will be stored in the cache in one of the W lines in set i mod (C/W). Associativity affects the attack in the following way. Since the array A has size C, the first step will still clean all lines of all sets of the cache. In the third step, we now need to check the W lines of the set where $S[y]$ could be present. We do this by reading the W elements $A[y]$, $A[y + C/W]$, $A[y + 2\ C/W]$, … and we record the sum of the W reading times.

Taking all improvements together, we obtain Algorithm 3 for the measurement phase of a cache timing attack. We stress again that the attacker should try to run these actions without interruption, in order to reduce the noise level.

Hardware performance counters

Modern processors include *hardware performance counters*, which contain precise information on the number of cache misses that have happened, and many more information. If we have the privileges required to read the data from the hardware performance counter, then we can simplify the measurement phase of any attack based on the number of cache misses.

Instead of relying on timings, which always contain noise, we get directly the number of cache misses. This implies that we will need much less measurements and encryptions in order to recover the key.

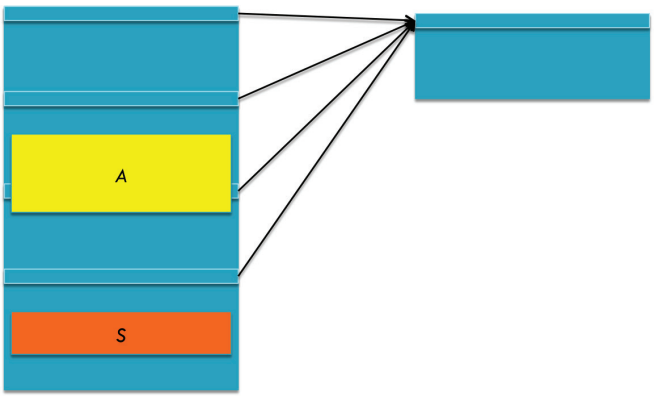


Figure 1. *Multiple addresses of the RAM (left) are mapped to each address of the cache (right). The arrays A and S are both in RAM*

Figure 2. *After Step 1, the array A fills the cache (left). After an encryption, some lines of S have kicked out some lines of A (right)*

Algorithms

Algorithm 1. *xtime: multiplication by x ("2")*

```
xtime(byte a) {
byte b;
If ((a & 0x80) == 0x80)
   then b = ((a & 0x7f) << 1) ^ 0x1b;
   else b = (a << 1);
return b;
}
```

Algorithm 2. *Determining a key byte from timing information*

```
determine_key_byte(double timing[256]) {
word k, kmax;
double sum[256];
for k = 0 to 255
   sum[k] = 0.0;
   for i = 0 to 255
      if (S[i xor k] & 0x80)
         then sum[k] = sum[k] + timing[i];
         else sum[k] = sum[k] - timing[i];
   end for
end for
// determine kmax: k with the largest sum[k]
...
output kmax;
}
```

Algorithm 3. *High-level description of the measurement phase of a cache timing attack*

```
collect() {
Run the program Prime.
Trigger the encryption of one block.
for y = 0, d, 2d, ..., 256 - d
   Start the timer.
   Read A[y], A[y + C/W], A[y + 2C/W], ..., A[y + C - C/W]
   Stop the timer and record the time.
end for
}
```

Conclusion

A naive implementation of a cryptographic algorithm often has an execution time that depends on the message input and the secret key. If an attacker can obtain reliable measurements of the execution times, then this will lead to side-channel attacks, which can break algorithms that are secure against purely mathematical cryptanalysis.

Even though a competent programmer can easily avoid the most obvious causes for variable execution times, advanced processors contain many features whose behavior is difficult to control and who may leak information. Leakage countermeasures often boil down to taking the advanced features out of action. For example, in order to achieve a constant execution time, a programmer may need to flush the cache after each encryption operation. Alternatively, the programmer can try to eliminate table lookups from the program. In the case of AES, this approach has been followed for example in [3].

On the other hand, the practical impact of timing attacks based on cache misses shouldn't be overestimated. Many of the attacks require detailed knowledge about the particular implementation of AES being used, and the attacker needs to be able to trigger en- or decryption operations. There are many situations where an attacker who is able to obtain timing information with the accuracy required for this attack, is already deep inside the system and can cause much more damage in more simple ways.

References

- [1] Specification for the Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, 2001.
- [2] Joan Daemen, Vincent Rijmen. The design of Rijndael: AES - the Advanced Encryption Standard, Springer 2002.
- [3] Emilia Käsper, Peter Schwabe. Faster and timing-attack resistant AES-GCM. CHES 2009, LNCS, vol. 5747, pp. 1-17, Springer, 2009.
- [4] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. CRYPTO 1996, LNCS, Vol. 1109, pp. 104-113, Springer, 1996.
- [5] Eran Tromer, Dag Arne Osvik, Adi Shamir. Efficient cache attacks on AES, and countermeasures. Journal of Cryptology, Vol. 23, No. 1, pp. 37-71, Springer, 2010.

VINCENT RIJMEN

is full professor at the University of Leuven, Dept. of Electrical Engineering/ESAT (Belgium) and part-time professor at the Graz University of Technology, Dept. IAIK (Austria). His research interests include design and cryptanalysis of ciphers and cryptographic hash functions.

Vincent is co-designer of the algorithm Rijndael, which has become the Advanced Encryption Standard (AES), standardized by the US National Institute for Standards and Technology (NIST) and used world-wide, e.g. in IPSec, SSL/TLS and other IT-security standards. Vincent is also the co-designer of the WHIRLPOOL cryptographic hash function, which made it into ISO/IEC 10118-3. Vincent authored numerous scientific publications in the field of symmetric cryptography. He was Program Chair of RFIDSec 2005, RFIDSec 2006, RFIDSec 2007, Program Co-chair of Fast Software Encryption 2002, Indocrypt 2008, Selected Areas in Cryptography 2009, and member of the program committee of many other international conferences in the field of cryptography.

Vincent's current research is centered around two topics. Firstly, the always ongoing evaluation of the security of AES against cryptanalysis. Secondly, the design of new techniques which should allow to construct implementations of AES (and other ciphers), wich are secured against side-channel attacks.

TIMING ATTACKS AGAINST RSA REVISITED

QI CHAI

The prevailing belief -- an information system is secure due to the employment of cryptographic functions that are mathematically strong -- can go wrong if adversaries does not play by the presumed rules. In fact, attacks may happen in completely unexpected ways such as compromising the cryptographic functions through measurements of the time they take to accomplish certain tasks, known as the *timing attack*. In this article, we exhibit instructive cases and examples of how to attack a weak class of implementations of the most popular public-key cryptographic algorithm, i.e., RSA, by making wise use of the runtime it reveals during operating. The lesson learnt is that the theoretic security of crypto algorithms should be examined in conjunction with the implementation security of them that may add another layer of complexity to the development of secure systems.

1 Introduction

1.1 What is RSA?

Before 1976, symmetric-key encryption, where the secret keys used by the encrypter, i.e., Alice, and the decrypter, i.e., Bob, are identical (or can be simply transformed from the one to the other), is the only known paradigm to protect data. To share the encrypted information with other parties, the secret key has to be distributed or delegated, which could be problematic, e.g., how to deliver the key effortlessly and secretly without introducing additional encryptions? how could one ensure that each of the key-holders will keep the key information privately from unauthorized parties especially in the long run?

This problem has been thoroughly solved thanks to Diffie and Hellman's breakthrough invention of the public-key cryptography, which, based upon the intractability of some computational hard problems, could accomplish the tasks like encryption and decryption using two different keys -- one is published and the other is private. Anyone

can use the public key to encrypt a message, while only the one with the knowledge of the private key can feasibly decode the message. Shortly after Diffie and Hellman's work, Rivest, Shamir and Adleman proposed a nice public-key scheme in [6], known as RSA, that leverages the difficulty of factorization of large integers. Nowadays, RSA has been considered as the most commonly used crypto primitive to enable confidentiality, integrity and authenticity of digital data, e.g., it has been deployed in majority commercial systems, it has been used by almost every web servers and browsers to secure web traffic and it is at the heart of electronic credit-card payment systems.

1.2 What is Timing Attack?

Albeit cryptography provides strong mathematical tools such as RSA to boost the system security, the actual security gain these tools offer largely depends on how they are used in real systems. A common pitfall is to over optimize the performance of a trusted cryptographic algorithm that yields a non-constant runtime as a (simple) function of some secret parameters. If so, by measuring and analyzing the time variations, the secret parameters can be learnt. In general, we call the attempts an adversary, i.e., Eve, made to compromise a crypto algorithm by observing the time it takes to process certain inputs as the *timing attack*.

Listing 1: A Common Login Function that Suffers Timing Attack

```
def login(username, password, users):
    if username in users:
        if password == users[username]:
            return True
    return False
```

A typical example is given here: when a user tries to login in, the web server authenticates him in two steps: (1) check whether the provided username exists in the system; (2) if so, check whether the provided password matches the record. While this snippet of code functions well under most circumstances, it threats the data privacy if the time spent on authenticating particular pairs of username and password has been measured maliciously. For example, Eve is able to tell whether a specific user is registered in this system since the login function takes longer to check an existed username than to check one not within the system. Furthermore, Eve could make use of the non-constant time of the comparison function "==" (if it is implemented by the “strcmp” function in C++) to find out the correct password bit by bit.

1.3 Our Mission

Since its initial publication, the RSA algorithm has been intensively cryptanalyzed due to its succinctness and robustness. Although twenty years of research have led to a number of fascinating attacks as summarized in [2], none of them is devastating.

Nevertheless, this is not equal to say that RSA is the armour with no chink. In fact, research indicates that, if not implemented properly, Eve is feasible to recover RSA's private key without directly breaking RSA. Timing attack against RSA, among other side-channel attacks, is the most effective and elegant method.

In this the article, we revisit and extend Kocher's work in [5] by exhibiting several real world timing attacks against a popular class of implementations of RSA algorithm. Our empirical study shows that the measured runtime can be used to probe the length of the private key in RSA and quickly discover the private key bit by bit, regardless of RSA's formidable mathematical strength.

2 How RSA Encryption/Decryption Works?

Before proceeding to the attacking methods, we have to go deep into the technical details of how RSA works mathematically and practically.

2.1 The Mathematical Idea

The mathematical idea of RSA can be presented as below.

Preparation:

1. Choose two large primes, say p and q, and calculate $N = p \times q$ and $\phi(N) = (p-1) \times (q-1)$.
2. Choose an integer $e < N$ that is relatively prime to $\phi(N)$.
3. Determine another integer d such that $e \times d \bmod \phi(N) = 1$.
4. Set (e, N) as the public key and d as the private key. Note that, p, q and $\phi(n)$ must be kept secretly as d can be calculated once they are leaked.

Encryption:

Let M, $0 < M < N$, be a plaintext or message, the corresponding ciphertext C is calculated by:

$$C = M^e \bmod N.$$

Decryption:

Let C, $0 \leq C < N$, be a ciphertext, the corresponding message M is calculated by:

$$M = C^d \bmod N = (M^e)^d \bmod N.$$

As of 2010, the largest known number factored by a general-purpose factoring algorithm was 768 bits long using paralleled computing. Henceforth, RSA keys

nowadays are typically 1024-bit long, while 2048-bit is a more recommended. Here we provide a toy example of RSA encryption and decryption.

Preparation:

1. Choose $p = 101$ and $q = 113$, and compute $N = p \times q = 11413$, and $\phi(N) = (p-1) \times (q-1) = 11200$.
2. Choose $e = 3533$ that is relatively prime to 11200.
3. Compute $d = e^{-1} \bmod \phi(N) = 6597$.
4. Set the public key as $(N, e) = (3533, 11200)$, and the private key as $d = 6597$.

Encryption: for plaintext $M = 9726$, compute $C = 9726^{3533} \bmod 11413 = 5761$.

Decryption: to decrypt 5761, compute $M = 5761^{6597} \bmod 11413 = 9726$.

Note that, by treating (N, d) as the public key and e as the private key, RSA encryption/decryption is turned to be RSA digital signature that is widely used in software distribution, financial transactions and other cases where it is crucial to detect forgery or tampering. Since RSA digital signature is only another way to use the RSA algorithm, our attacks as shown later is naturally applicable to RSA digital signature as well.

2.2 The Practical Realization

Since RSA deals with modular exponentiations of very large integers, e.g., 1024-bit integers, the naive implementation would be prohibitively slow. As a consequence, how to make the modular exponentiation computationally efficient becomes a vital issue for this algorithm. There are few known ways to achieve this goal such as square-and-multiply, sliding window method, Chinese Remainder Theorem and Karatsuba multiplication. Among them, square-and-multiply is the most universal strategy, which could start the processing either with the most significant bit (MSB) of the exponent -- call it *left to right square-and-multiply* -- or with the least significant bit (LSB) of the exponent -- call it *right to left square-and-multiply*. The Python implementations of both flavors are shown below.

Listing 2: left to right square and multiply -- star with the MSB

```
def decryption(C, d, N, round):
    M = 1
    for i in xrange(round):
        M = (M*M)%N              #square
        if (d>>(round-i))&0x01 == 1:
            M = (M*C)%N          #multiply
    return M
```

Listing 3: right to left square and multiply -- star with the LSB

```
def decryption(C, d, N, round):
    M = 1
    z = C
    for i in xrange(round):
        if (d>>i)&0x01 == 1:
            M = (M*z)%N         #multiply
        z = (z*z)%N              #square
    return M
```

To be specific, for the right to left square-and-multiply, let $d = (d_{w-1}d_{w-2}...d_0)$ be the binary representation of the private key or the secret exponent d. To compute $M = C^d \bmod N$, the algorithm actually computes $M = \prod_{i=0}^{w-1} C^{2^i \cdot d_i} \bmod N$, meaning that in each iteration the variable M "collects" the appropriate powers in the set to construct $M = C^d \bmod N$. As both derivatives of the square-and-multiply can be attacked in a similar way, we concentrate on attacking the RSA decryption implemented by the right to left square-and-multiply.

Here we provide another small example of the right to left square-and-multiply to make it intuitive. To compute $10^{13} \bmod 35$ (where the binary representation of 13 is 1101), the algorithm conducts the following steps one by one. Through the example, a sharp reader may sense an early warning of the timing attacking due to a missed operation in the second iteration.

Table 1: An Example of the Right to Left Square-and-multiply

Iteration	$M = (M \times z) \bmod N$	$z = (z \times z) \bmod N$
1	$M = 1 \times 10 = 10 \bmod 35$	$z = 10^2 = 30 \bmod 35$
2		$z = 30^2 = 25 \bmod 35$
3	$M = 10 \times 25 = 5 \bmod 35$	$z = 25^2 = 30 \bmod 35$
4	$M = 5 \times 30 = 10 \bmod 35$	$z = 30^2 = 25 \bmod 35$

3 Attacking RSA through Timing

3.1 Attacking Scenario

To make our attacks more instructive and concise, we consider a "local" attacking scenario such that Eve is able to access the *target device*, e.g., a server or a tamper-resistant smartcard, that stores the private key and runs RSA encryption and decryption (implemented by the right to left square-and-multiply) when stimulated, and Eve has a physical clone of the target device, e.g., another server of the same model or another smartcard. In addition, we assume that Eve is able to measure the time spent on the RSA decryption on the target device and any other operations on the cloned device that do not request secret parameters.

The target device and the cloned device we used for our empirical study are a PC

powered by a 2.66 GHz Intel dual-core i7 CPU and runs Ubuntu 10.04 TLS and Python 2.6. Compared with attacks performed on dedicated hardware, e.g., smartcards [4], our testbed is more budget-friendly but provides less accuracy due to the undesired parallelism in the CPU, the noisy background threads in the OS and the limited precision of the timing mechanism in software. However, with countermeasures such as disabling the multi-core and hyper-threads functionalities on the CPU, turning off GNOME, and repeating each measurement reasonable number of times, we demonstrate that it is feasible to mount timing attacks in a pure software environment that only offers reasonably accurate timing measurements.

3.2 An Amateur's Try: Probing the Key Length

We start with a very basic attack evaluating the key strength of the RSA algorithm. It is quite intuitive that the time consumed by the decryption is dominated by the number of iterations or for-loops executed, which is linear in the length of the binary representation of the private key. Consequently, to determine the key length via timing, Eve mounts the following:

1. On the cloned device, Eve runs the decryption enough times with a random ciphertext C and several length-varying exponents d's that are randomly selected as well. Eve records the time required to perform each decryption, denoted as T', with the corresponding C and d'.
2. Eve then queries the decryption of C on the target device and records the time it costs, denoted as T.
3. Eve searches in T's for the one closest to T. If success, Eve could roughly assert that the length of the secret key d is equal to the length of a particular d' (assuming the corresponding T' is the one closest to T).
4. The above steps can be repeated arbitrary number of times and the most suggested length is the correct length of the private key.

We instantiated this conceptual attack through the following Python code, where we select N to be a secure 1026-bit integer, M to be a 258-bit integer, and d to be a random integer with its length ranging from 128-bit to 1024-bit. The times taken to perform 1000 decryptions are pictorially depicted in Fig. 1. As we expected, the time spent is strongly correlated to the length of the key, from which Eve is capable of collecting more information regarding the secret key that might further optimize his attacking strategy. For instance, if d is selected to be a small integer (surprisingly, this was a common pitfall for practitioners intending to improve the decrypter's performance) less than 80-bit, a simple brute force search is applicable. In addition, if the key length is larger than 80-bit but less than $d < \frac{1}{3}N^{1/4}$, Wiener's attack in [7] also breaks the entire RSA algorithm.

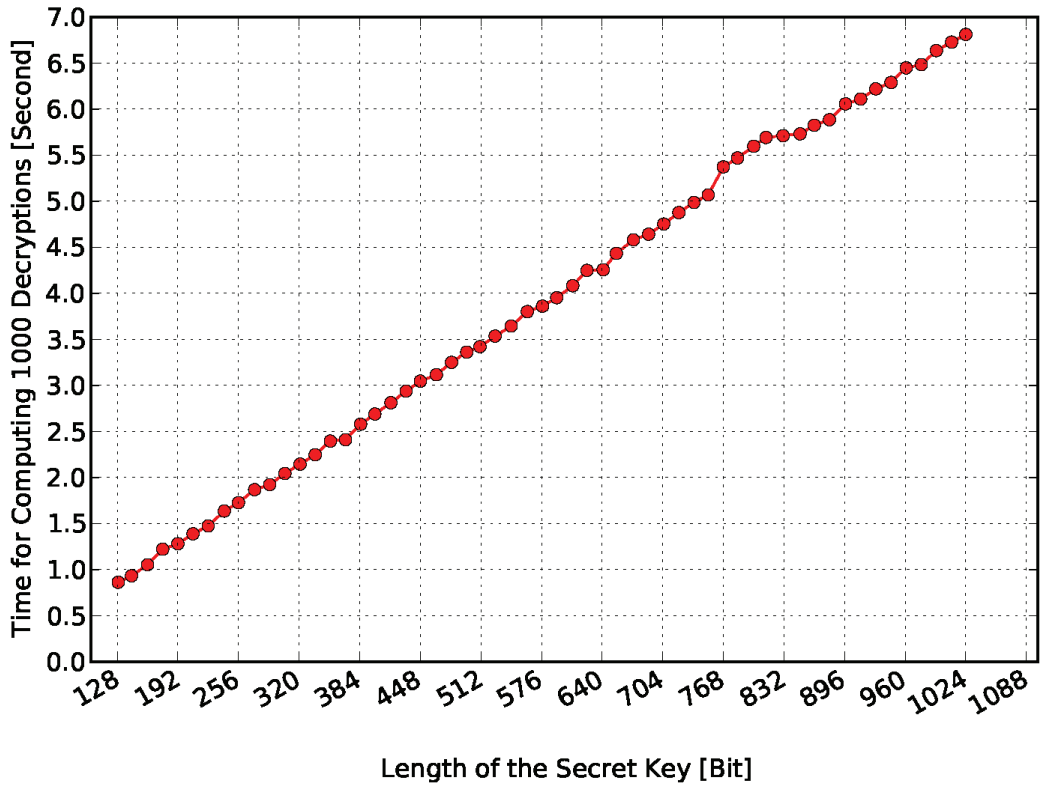


Figure 1: Probing the Key Length Through Timing

3.3 A Step Further: Compromising Each Iteration

Besides the non-constant time as introduced by the varying number of for-loops, one may notice, from our previous examples, that each iteration of the square-and-multiply may consume different amount of time, due to three possible factors:

1. with or without computing $M = (M \times z) \bmod N$ depending on whether d_i is 1 or 0;
2. modular multiplication of different integers takes different times;
3. squaring of different integers takes different times.

To examine these intuitions, we measured the times spent on squaring and the times spent on square and multiply respectively during each of the 256 iterations in an RSA decryption of a random ciphertext using the following snippet of code. Our results are presented in Fig. 2. From this figure, one could easily be convinced that our hypotheses (1) and (2) are true while (3) is not -- actually, the squaring operation takes almost a constant time regardless of the input. By scanning Fig. 2, we could even read out of the bits of d in each iteration, i.e., the lower blue bars suggest d_i s in these iterations are very likely to be 0 and vice versa.

Listing 4: Measurement of Each Iteration of RSA Decryption

```
C = 0x00c59068b22b162d6dd1a71b40b594d3163eeeb2b8ec2a1905a95c27d9461bd59L
d = 0x008d0086c444f02b8c0822d01fe64216c1a19ef9ea4d07b7ce57587dd668dde979L
N = 0x00e00d9842b6c6451314a4a8536e9c7667505e135350cde8268ce9c2812673435ea9b2b
    749fe51b98c6c311d63e02bfb458c9d075155d7ccb72144bc34023b3887L
A = list([]) #time for a (possible) multiply
B = list([]) #time for a square

def decryption(C, d, N, round):
    M = 1
    z = C
    for i in xrange(round):
        start = time()
        if (d>>i)&0x01 == 1:
            M = (M*z)%N
        A.append(time() - start)
        z = (z*z)%N
        B.append(time() - start)
    return M
```

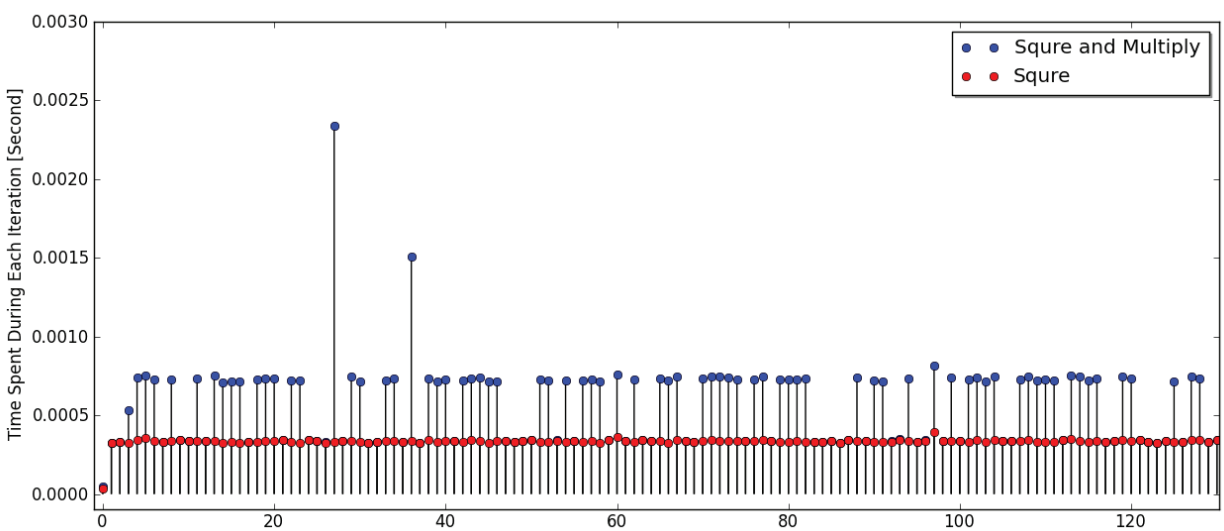


Figure 2: Runtime of First 128 Iterations in an RSA Decryption (The LSB 128-bit of the secret component is "10100001100111101111100111101010010011010000011110110111110011100101011101011000011111011101011001101000110111011110100101111001" that can even be read out from lengths of blue bars.)

Though the attack impractically requires the attacker's capability to measure each iteration of an RSA decryption, it uncovers two significant clues, i.e., (1) and (2), that lead to a devastating timing attack to recover d_i in each iteration, as shown in the next.

3.4 A Real Attack: Whole Key Recovery

Now, we are ready to perform a real attack to recover the private key bit by bit starting from the second LSB of the secret component (as the first LSB could invariably be 1). The idea follows from the restatement of our previous observations (1) and (2): for a specific ciphertext, if both the first two iterations and the entire decryption take much longer

(shorter resp.) than usual, it is highly probable that the modular multiplication is involved in the second iteration, which further suggests that $d_1 = 1$; if they do not share such a "trend", it is very likely $d_1 = 0$. So as to attack, Eve has to emulate and measure the first and second iterations on the cloned device, while measure the decryption on the target device. Since Eve has to obtain the usual or average case for comparison, this method is statistic in nature. After the recovery of d_1, Eve is able to emulate one more iteration on the cloned device to attack d_2 and so on. In what follows, we explain, using tools from probability theory, how Eve recognizes and learns this "trend" during the attack.

For convenience, let us say the decryption of a ciphertext C_i encompasses w iterations and let τ_j, $0 \le j < w$, denote the time spent on the j th iteration of decryption. As a result, the overall time spent on the entire decryption, denoted as T_i, is the sum of τ_j s, measurement errors and the loop overhead (denote the later two as e), i.e.,

$$T_i = e + \sum_{j=0}^{w-1} \tau_j .$$

Since the modular multiplication times are effectively independent from each other and from the measurement error and the loop overhead, the variance of T_i over all observed ciphertexts is expected to be

$$var(T_i) = var(e) + w \cdot var(\tau).$$

In addition, with the knowledge of $(d_{b-2} \ldots d_0)$, Eve guesses d_{b-1} and emulates the first b iterations with C_i on the cloned device to obtain the runtime:

- If Eve guesses correctly, the runtime of the emulation is $(\tau_0 + \tau_1 + \ldots + \tau_{b-1})$ and

$$X = T_i - (\tau_0 + \tau_1 + \ldots + \tau_{b-1}) = e + \sum_{j=b}^{w-1} \tau_j .$$

Therefore, the variance of X is $var(X) = var(e) + (w - b) \cdot var(\tau)$.

- On the contrary, if Eve guesses incorrectly, the runtime of the emulation is $(\tau_0 + \tau_1 + \ldots + \tau_{b-2} + \tau'_{b-1})$ and

$$Y = T_i - (\tau_0 + \tau_1 + \ldots + \tau_{b-2} + \tau'_{b-1}) = e + \sum_{b-1}^{w-1} \tau_j - \tau'_{b-1}.$$

Therefore, the variance of Y is $var(Y) = var(e) + (w - b + 2) \cdot var(\tau)$.

As a consequence, when Eve makes an erroneous guess of d_{b-1}, the variance of the difference between the runtime of the decryption on the target device and the runtime of the emulated iterations on the cloned device is larger, which is apparently a distinguisher that tells Eve the true value of d_{b-1}. This attacking procedure begins with

$b=2$ and ends with $b=w-1$ and recovers one bit of d per execution.

With 256 calls of the Python code given below, we mounted an attack to recover the 256-bit private key in an RSA instance using only 64 random ciphertexts. The partial results, e.g., while attacking the 4 th, the 32 th, the 96 th and the 256 th iterations, are listed numerically in Table 2 and exhibited graphically from Fig. 3 to Fig. 6.

Listing 5: Recovery RSA Private Key Real Via Timing

```
b = 96
C = 0
d = 0x008d0086c444f02b8c0822d01fe64216c1a19ef9ea4d07b7ce57587dd668dde979L
N = 0x00e00d9842b6c6451314a4a8536e9c7667505e135350cde8268ce9c2812673435ea9
b2b749fe51b98c6c311d63e02bfb458c9d075155d7ccb72144bc34023b3887L
guess_d = ( d&((1<< (b-1))-1) )| ((1<< (b-1)) ^ d&(1<< (b-1)))
# (b-1) LSB are correct while the bth bit is incorrect
myT, myT0, myT1 =   list([]), list([]), list([])

def measure(Iteration):

    #decryption on the target device
    t = Timer("decryption(C, d, N, 256)", "from __main__ import decryption, C, d, N")
    S = t.repeat(1,100)
    cnt = 0
    for s in S:
        cnt += s
    myT.append(   cnt   )

    #emulation on the cloned device with the wrong guess of d_{b-1}
    t = Timer("decryption(C, guess_d, N, b)", "from __main__ import
    decryption, C, guess_d, N, b")
    S = t.repeat(1, 100)
    cnt = 0
    for s in S:
        cnt += s
    myT0.append( myT[-1]-cnt )

    #emulation on the cloned device with the right guess of d_{b-1}
    t = Timer("decryption(C, d, N, b)", "from __main__ import
    decryption, C, d, N, b")
    S = t.repeat(1, 100)
    cnt = 0
    for s in S:
        cnt += s
    myT1.append( myT[-1]-cnt )

if __name__ == '__main__':
    for i in xrange(64):
        C = randint(0, N)
        measure(i)
    print var(myT), var(myT0), var(myT1)
```

In Table 2, one can see that the variance of the difference between the runtime of the decryption and the runtime of the emulation is clearly smaller as long as d_{b-1} is correct, as analyzed. It is also worth pointing out that both variances $var(X)$ and $var(Y)$ are decreasing with the increasing in the number of iterations. This is mainly because, with the correct guess of $(d_{b-2}...d_0)$, $(\tau_0+\tau_1+...+\tau_{b-1})$ has been properly removed from T_i and the remaining randomness has been diminished.

Fig. 3 to Fig. 6, by plotting the normalized times spent on decryption and emulation of 4, 32, 96, 256 iterations, present the same results in a more intuitive way. For instance, when more bits of d are guessed correctly, the runtime of the decryption

and the runtime of the emulation are more correlated.

Table 2: Some Variance as Obtained in Our Timing Attack

Iteration b	$var(Y)$ (incorrect guess of d_{b-1})	$var(X)$ (correct guess of d_{b-1})
4	$7.66838596239 \cdot 10^{-5}$	$7.64611866879 \cdot 10^{-5}$
32	$5.60113336672 \cdot 10^{-5}$	$5.36986543831 \cdot 10^{-5}$
96	$3.19115063296 \cdot 10^{-5}$	$2.47564144942 \cdot 10^{-5}$
256	$2.28249098881 \cdot 10^{-5}$	$1.30603068784 \cdot 10^{-5}$

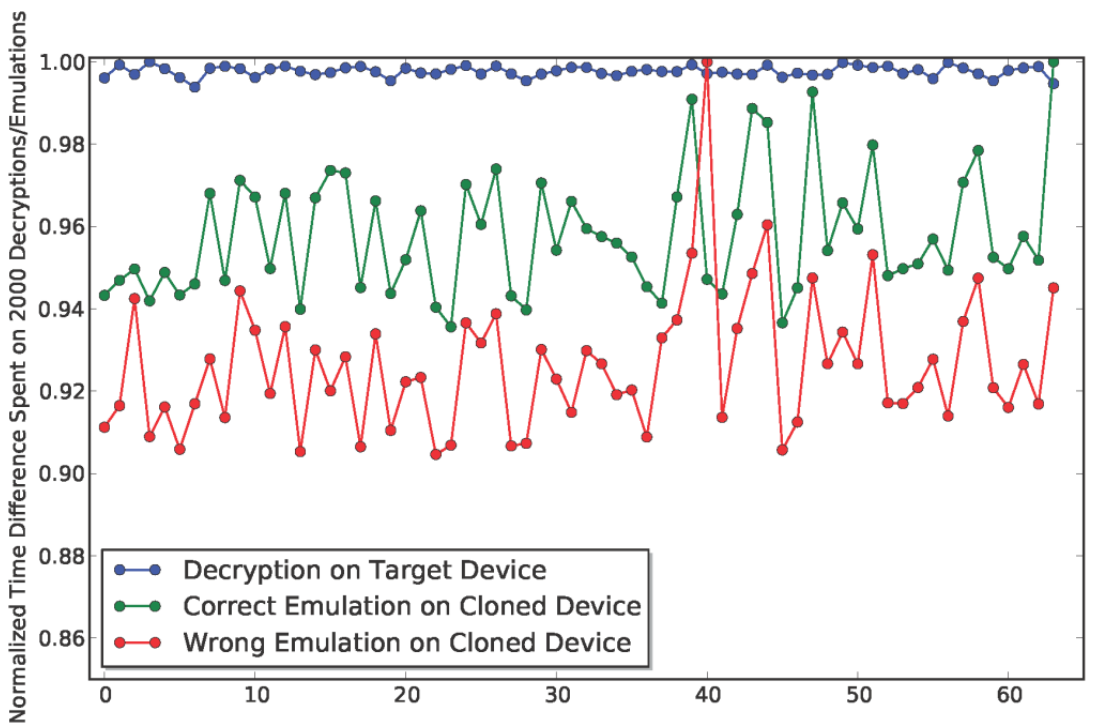


Figure 3: Normalized Times Spent on Decryption and Emulation of 4 Iterations

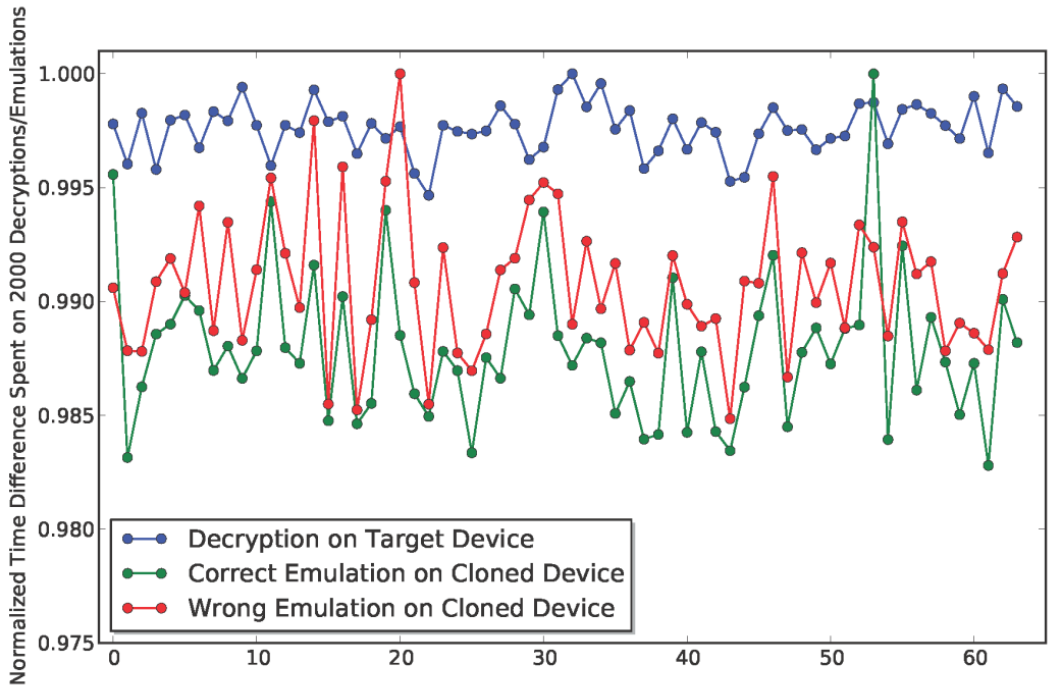


Figure 4: Normalized Times Spent on Decryption and Emulation of 32 Iterations

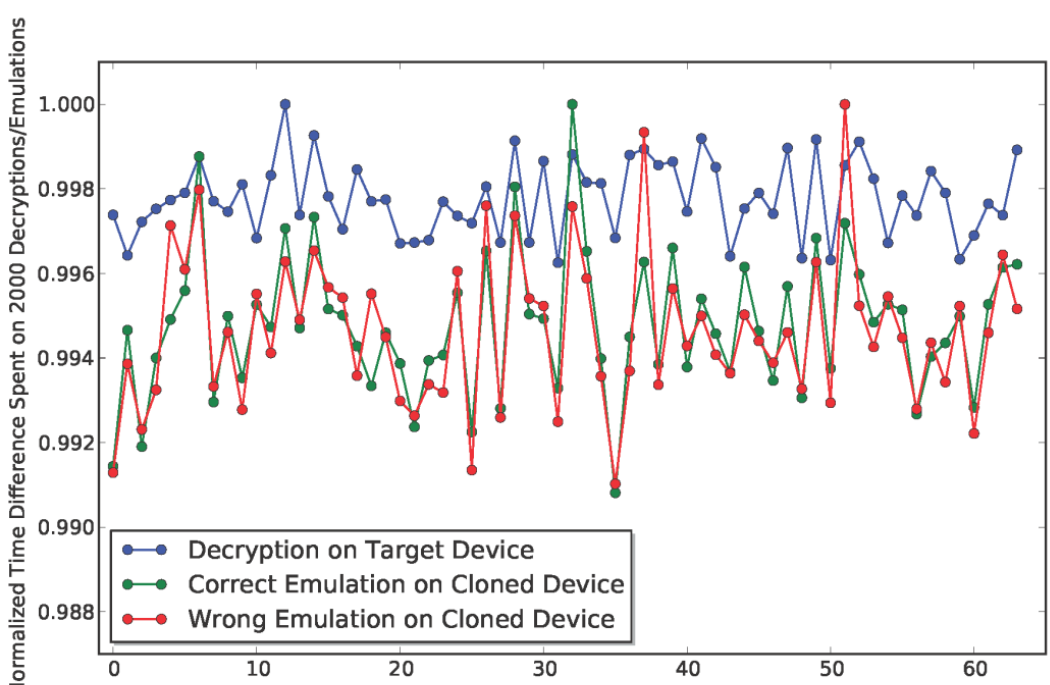


Figure 5: Normalized Times Spent on Decryption and Emulation of 96 Iterations

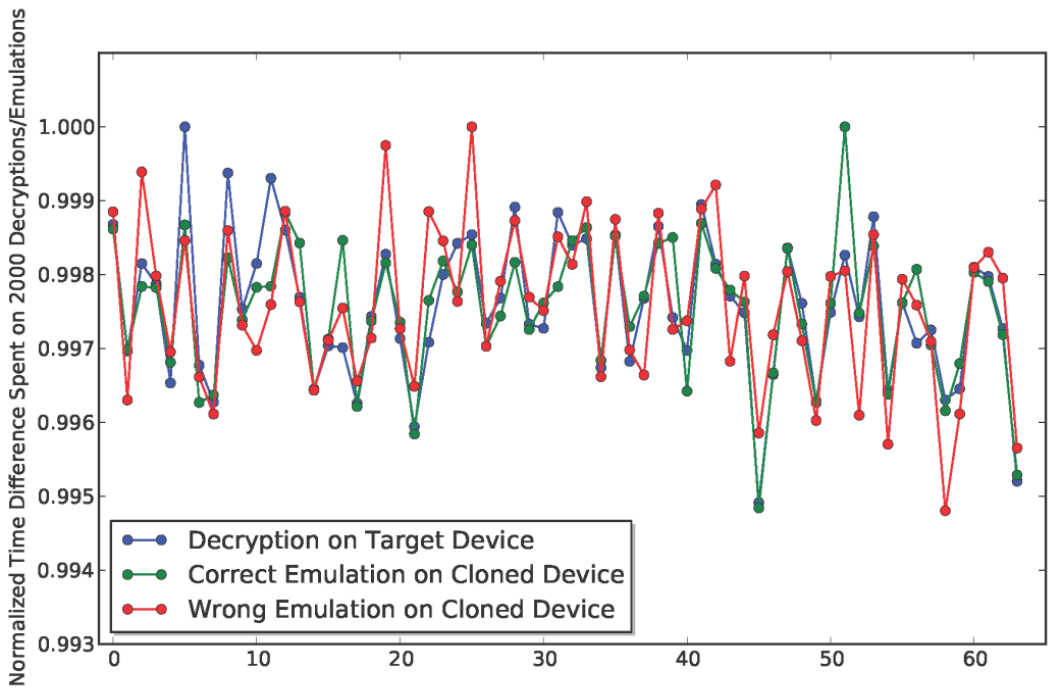


Figure 6: Normalized Times Spent on Decryption and Emulation of 256 Iterations

4 Possible Defenses

As the saying goes, it is better for the doer to undo what he has done. To prevent the leakage of the timing characteristics as a function of the secret component, defensive codes must be added. In the case of square-and-multiply implementation of RSA, there are two common remedies: time compensation and blinding.

4.1 Time Compensation

To make sure that every iteration takes almost the same amount of time regardless of value of d_i, the simplest is to add a proper delay to the "else" branch. The following code demonstrates this idea. However, the computation of $(M \times z) \bmod N$ is apparently wasted providing $d_i = 0$. By assuming around half bits of d are zero, the time-compensated square-and-multiply takes approximately twice the time (and twice the energy) to finish the same computation, compared with the original implementation.

Listing 6: Time-Compensated right to left square and multiply

```
def decryption(C, d, N, round):
    M = 1
    z = C
    for i in xrange(round):
        tmp = (M*z)%N            #multiply
        if (d>>i)&0x01 == 1:
            M = tmp
```

```
        else:
            tmp = tmp
        z = (z*z)%N             #square
    return M
```

4.2 RSA Blinding

The second approach, due to Rivest, is more preferred, where the randomness is introduced into the RSA operations to make the runtime uncorrelated to the secret component. This method is named as *RSA blinding*. For instance, prior to the encryption of M, Alice picks up a random integer r, $0 < r < N$, and computes $C' = (M \times r)^e \bmod N$. Bob then applies d to C' and obtains $M' = (C')^d \bmod N$ and $M = (M' \times r^{-1}) \bmod N$. With this approach, a random value multiplied to the variable M in each iteration decouples the runtime of the algorithm from the secret component. Unlike the first method, blinding only incurs a small performance penalty in the range of 2% to 10%.

5 Conclusion

Timing attack, exploiting the timing variations in cryptographic primitives, is the sword of Damocles hanging over the head of the implementers of secure systems. To stress the importance of the implementation security of crypto algorithms and to stimulate research towards this direction, we provide an instructive case study in this article and show that if the RSA decryption can be timed reasonably accurately in a software environment, analysis can be applied to recover the private key involved in the computations. The principles we introduced through a local attacking scenario are universal and can be applied to attack RSA remotely, e.g., [3].

Besides, it is worth mentioning that similar timing attacks can be applied to other weak implementations of RSA algorithm where either Montgomery reduction or Chinese Remainder Theorem is used. Moreover, timing attacks are quite powerful that are not restricted to work with public-key algorithms. In reality, block ciphers and stream ciphers are even more suffered, e.g., cache-timing attacks on AES [1].

References

[1] D.J. Bernstein, Cache-timing attacks on AES, preprint available from *http://cr.yp.to/antiforgery/cachetiming-20050414.pdf*, pp. 1--37, 2005.
[2] D. Boneh, Twenty years of attacks on the RSA cryptosystem, *Notices of the American Mathematical Society (AMS)*, vol. 46, no. 2, pp. 203--213, 1999.
[3] D. Brumley and D. Boneh, Remote timing attacks are practical, *Computer Networks*, vol. 48, no. 5, pp. 701--716, 2005.
[4] J.F. Dhem, F. Koeune, P.A. Leroux, P. Mestré, J.J. Quisquater, and J.L. Willems, A practical implementation of the timing attack, *Smart Card Research and Applications*, pp. 167--182, 2000.
[5] P. Kocher, Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems *Advances in Cryptology, CRYPTO'96*, pp.104--113, 1996.
[6] R.L. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, vol. 21, no. 2, pp. 120--126, 1978.
[7] M. Wiener, Cryptanalysis of short RSA secret exponents, *IEEE Transaction on Information Theory*, vol. 36, no. 3, pp. 553--558, 1990.

Short Biography

Qi Chai received the B.S. degree in 2005 and the M.S. degree in 2007 in electrical engineer from Beijing Institute of Technology, China. He received the Ph.D. degree in January 2012 in electrical and computer engineering from University of Waterloo, Canada. He is now a senior researcher, member of CACR (Centre for Applied Cryptographic Research) and IACR (International Association for Cryptologic Research). Dr. Chai is an information security generalist and currently works on the lightweight cryptography, wireless security, cloud computing security and intrusion detection.

inj3ct0r

if you'll hacked us
we'll pay you 10K $
http://1337day.com/

Exploit database separated by exploit type
(local, remote, DoS, Poc, etc.)

VARIANT PSEUDO-RANDOM NUMBER GENERATOR

WEIZHONG YANG, JEFFREY ZHI J. ZHENG

Abstract – Variant Pseudo-Random Number Generator (VPRNG) based on the Variant Logic framework – an extension of Cellular Automata (CA) - is proposed to construct a PRNG. A list of classical methods on PRNG, BBS, ANSI X9.17 and DES were used in comparison under the NIST Statistical Test Suite, the measurement results show that the VPRNG can produce a better pseudo-random number series in most cases than the compared models.

Introduction

The security of cryptographic systems depends on secret data that is known for authorized persons but unknown and unpredictable to others [1,3,5]. To meet this unpredictability, some randomness mechanisms are required [4,8]. Quality in Pseudo Random Number Generation PRNG is required for security in both stream cipher and block cipher applications [3,4,7,8], and lack of quality generally provides attack vulnerabilities [3,8]. From a practical viewpoint, some key properties for good randomness mechanisms are essentially important such as period length, efficiency and ease of implementation [1,3,7].

For some PRNGs, the period length can be calculated without walking through the whole period. For a PRNG internal state contains *n* bits, Linear Feedback Shift Registers LFSRs usually have periods of exactly $2^n - 1$ Normally systems based on essential Boolean operations are in higher performance and better efficiency in hardware or firmware implementations than compared systems based on arithmetical operations [3,8,9].

The PRNG system is particularly attractive to attackers because it is typically a single isolated component easy to be located in environment. Different cryptanalytic attack technologies are applied to PRNG mechanisms such as guessing of seed, timing attacks on state advance function, output generation functions, forward and backward tracking attacks [2,5-7].

To secure wider applications, it is a challenge task to design and implement a proper PRNG to have a list of superior properties [1-9].

In this paper, a new PRNG system – Variant Pseudo-Random Number Generator VPRNG – based on [12] is proposed. Combining different technology [10-12] with additional extensions, this system can provide extreme longer circular properties for a *n* bit state, VPRNG have a basic period of $2^n \times 2^{2n}$ bits for a configuration and also a huge variation space of $2^n \times 2^{2n}$ configurations available to use efficient tabular operations with significantly superior behaviors in comparison to other PRNG results. The system of VPRNG is described in section II-VI.

Variant Pseudo Random Number Generator

The architecture

The architecture of Variant Pseudo-Random Number Generator VPRNG is shown in Figure 1.

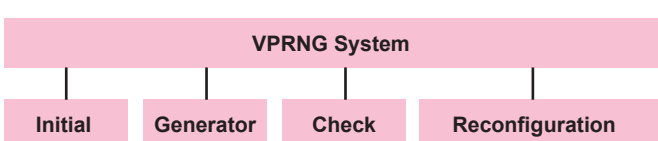


Figure 1. *The Architecture of Variant PRNG*

In the architecture, there are four modules: Initial, Generator, Check and Reconfiguration. Further detailed descriptions on various parameters are discussed in Sections III-VI respectively.

In the Initial module, it provides a list of initial arguments for the Generator, such as the Cell Serial X as a seed, the rule R, the complement Δ and the permutation P operators respectively.

Using the rule R, the Generate module selects a complementary rule Δ and a permutation rule P and works on Cell sequence to generate new Cell series. Under a given configuration, this module can provide a non-repeat sequence with a length of $2^n \times 2^{2n}$ bits [12].

In the check module, it focuses on checking the current Cell series in which whether its content has or not to be repeated. If it does not repeat, then it will be passed.

In the Reconfiguration module, it reconfigures arguments of the Cell Serial X, the rule R, the complement Δ and the permutation P into a new configuration. Under different configuration, this module can provide a huge variation space with configurations for the construction $2^n \times 2^{2n}$ [10-11].

Three core modules: Initial, Check and Reconfiguration are new modules extending to [12] method. This set of additional modules provides maximal flexibility to generate good randomness properties for the proposed mechanism to meet potential requirements on various applications.

The Procedure

The core procedure of VPRNG is processed as follows. Their input, output and process are listed.

Input: A N-bit vector X_N^0, a rule R, a permutation P, a complement Δ and COUNT (i=0);

Process: the whole procedure of VPRNG is shown in Figure 2. The detailed descriptions of each operation are explained in Sections III-VI respectivelv.

Output: A set of N-bit vectors $\{X_N^i\}_{i=1}^{COUNT}$

The main symbols and concepts of the VPRNG operations are described in Sections III-VI and further explanantions on their foundation are described in [10-12] respectively. In Section III, classical truth tables are discussed, in Section IV, variant representations are illustrated, in Section V, permutation properties are discussed and in Section VI, joined permutation and complementary operations, a new level of configuration space is constructed (Figure 2).

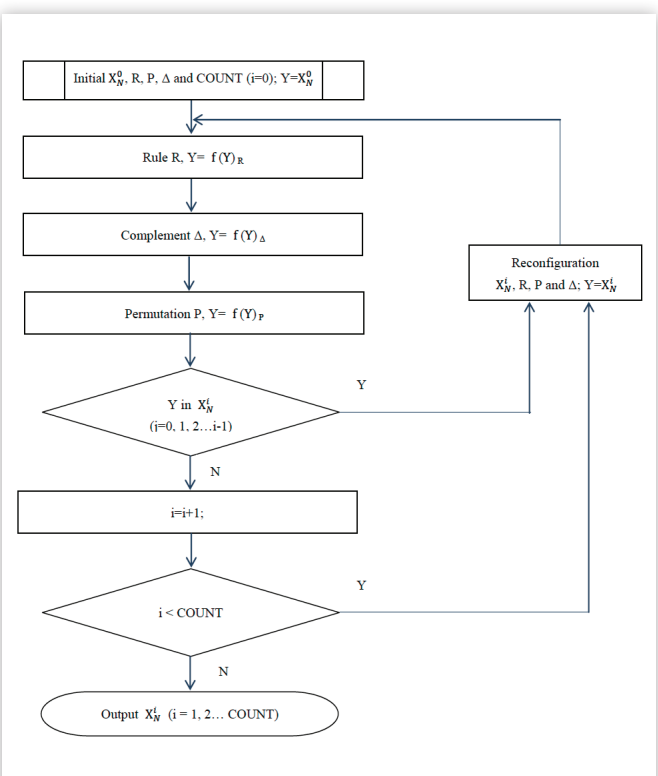


Figure 2. *Procedure of the system*

Truth table in classical boolean logic

The truth-table plays a vital role in traditional logic construction; It provides a static structure, using Yes | No (1|0) to indicate possible conditions from variables, states to any function. The proposed framework includes traditional logic function space and the truth-table representation of the space.

Basic definitions

$$X = X_{N-1}X_{N-2}...X_j...X_1X_0,\ Y = Y_{N-1}Y_{N-2}...Y_j...Y_1Y_0$$

$$X_i, Y_i \in B_2 = \{0, 1\},\ 0 \le i < N$$

$$f: X \to Y;\ Y = f(X);\ X, Y \in B_2^N = \{0, 1\}^N$$

An example of a transform: the sequence X = 0001110100, N = 10 is an input for a function operation f , the output is a sequence of the same length Y = 1101011001; X, Y ∈ B_2^{10}

Definition 2.1 Let [...X_j...] be a n bit structure as a kernel form:

$$[...X_j...] = x_{n-1}x_{n-2}...x_i...x_1x_0 = x \quad (2)$$

$$0 \le i < n,\ 0 \le j < N,\ x \in B_2^n$$

where $X_j = x_i$ is a corresponding position.

$$Y_j = f([...X_j...]) = f(x_{n-1}x_{n-2}...x_i...x_1x_0) = f(x) \quad (3)$$

In Boolean logic, n variables in a kernel form correspond to a full truth table with 2^n× entries. The I-th meta-state $0 \le I < 2^n$ has n bit number to occupy the I-th column position, the J-th function T(J) has the J-th row with 2^n bits $0 \le J < 2^{2^n}$, the function value of the I-th entry is determined by $T(J)_I$. The full table can be represented as follows:

From this type of tables, it is feasible to establish accessing method to look up corresponding table values.

Method 2.1: Process Method of Truth Table:

Input: x : n variables in a { 0, 1 } sequence, J: selected function number

$0 \le I < 2^n$	S_{2^n-1}	...	S_I	...	S_1	S_0
$I_{n-1}...I_i...I_0$	1...1...1	...	$I_{n-1}...I_i...I_0$	...	0...0...1	0...0...0
$0 \le J < 2^{2^n}$	J_{2^n-1}	...	J_I	...	J_1	J_0
$T(0)$	0	...	0	...	0	0
$T(1)$	0	...	0	...	0	1
$T(2)$	0	...	0	...	1	0
...			...			
$T(J)$	J_{2^n-1}	...	J_I	...	J_1	J_0
...			...			
$T(2^{2^n}-2)$	1	...	1	...	1	0
$T(2^{2^n}-1)$	1	...	1	...	1	1

Figure 3. *Truth Table 1*

Process: Using the input sequence x, the meta-state number I is to select the I-th column of function T(J)

Output: Return $T(J)_I$'s value (1 for true and 0 for false) as output.

Variant representations

Basic representation

Cellular Automata – CA uses a recursive mechanism to represent a given function with a time direction. Dynamic properties of CA can be supported in further expansions. In a one dimensional form of CA, a N length binary sequence is:

$$X = X_{N-1}X_{N-2}...X_j...X_1X_0,\ 0 \le j < N,\ Xj \in \{0, 1\} = B_2$$

For a given function f , the output sequence is defined:

$$f: X \to Y,\ Y = f(X),$$

$$Y = Y_{N-1}Y_{N-2}...Y_j...Y_1Y_0,\ 0 \le j < N,\ Yj \in B_2$$

It is feasible to use a moving window with a fixed length n to separate X into a local kernel in length n. The kernel can be presented as

$$[...X_j...] = x_{n-1}...x_i...x_0,\ x_i \in B_2$$

For a given function f

$$y = f(x_{n-1}...x_i...x_0)$$

It is necessary to assign a certain position i in the kernel for special care to associated with j position of both sequences. All above relations are exactly same as traditional Boolean equation with n variables.

It is possible to distinguish current time and next time sequences, following equations relevant to cellular automata can be identified:

$$y = f(x_{n-1}...x_i...x_0) = f([...X_j...]) = Y_j$$

$$\text{or } X_j = X_j^{t-1},\ Y_j = X_j^t \text{ i.e.}$$

$$f: X_j^{t-1} \to X_j^t,\ X_j^{t-1}, X_j^t \in B_2 \quad (4)$$

Four variation forms

Time direction is a significant property to distinguish a Cellular Automata logic function from a traditional logic function. Considering $f: X_j^{t-1} \to X_j^t$ for any function of Boolean logic system to analyze their variation properties, it is normal to have following proposition.

Proposition 3.1 For any $f: X_j^{t-1} \to X_j^t$ transformation, four forms of transforming classes are identified: TA: 0 → 0, TB: 0 → 1, TC: 1 → 0, TD: 1 → 1.

Proof: Xj, Yj are 0-1 variables, only four classes listed are possible.

Definition 3.1 Four Transforming forms are corresponding to following sets: TA: Invariant-valued class for 0 value, TB: Variant-valued class for 0 value, TC: Variant-valued class for 1 value, TD: Invariant-valued class for 1 value. Under such definition, following proposition can be established.

Proposition 3.2 Using four classes of transformation, four variant operations are defined.

Type	$X_j \to Y_j$		Truth	Variant	Invariant	False
TA	0	0	0	0	1	1
TB	0	1	1	1	0	0
TC	1	0	0	1	0	1
TD	1	1	1	0	1	0

Figure 4. *Truth/False Values*

Proof: Truth(False) values are determined by $Y_j(\neg Y_j)$ and Variant(Invariant) values are determined by {TB, TC} for 1(0) and {TA, TD} for 0(1) respectively.

Permutation invariants

Permutation operation

From an accessing viewpoint, many invariant properties can be observed from table operation.

Proposition 4.1 Under sequential mapping in sequential order of Method 2.1, there is T(J) =J.

Proof: The relevant output entries of T(J) are mapped to the binary number J having 2^n bits:

$$T(J) = T(S_{2n-1}(J_{2n-1}))...T(S_I(J_I))...T(S_0(J_0))$$

$$= T(J)_{2n-1}...T(J)_I...T(J)_0 = J \in B_2^{2^n} \quad (5)$$

$$T(J)_I = T(S_I(J_I)) = J_I \in B_2;\ 0 \le I \le 2^n,\ 0 \le J < 2^{2^n}$$

It is possible to apply permutation operation on the table to generate a transformed table following a certain rule.

Definition 4.1 For any n binary logic variables, let Ω(N) be a symmetric group with N elements and P be a permutation operator, $P \in \Omega(2^n)$, then for any J, ∃K, J, K ∈, $B_2^{2^n}$ P(T(J)) = K, $0 \le J, K < 2^{2^n}$, the following permutation can be represented in Truth Table form:

$$P: J \to K$$

$$P(T(J)) = P(T(S_{2n-1}(J_{2n-1})))...P(T(S_I(J_I)))...P(T(S_0(J_0)))$$

$$= P(T(J)2^n - 1)...P(T(J)_I)...P(T(J)_0)$$

$$= K_{2n-1}...K_I...K_0 = K \in B_2^{2^n}$$

$$P(T(J)_I) = P(T(S_I(J_I))) = T(S_{P(I)}(J_{P(I)}))$$

$$= T(J)_{P(I)} = J_{P(I)} = K_I \in B_2$$

$$0 \le I < 2^n,\ 0 \le J, K < 2^{2^n},\ P \in \Omega(2^n)$$

Proposition 4.2 The Truth Table under permutation operation on meta states can generate $2^n!$ Sequences for length of integers.

Proof: For any $P \in \Omega(2^n)$, 2^n are independent, it is composed of $\Omega(2^n)$ elements.

For the one-variable condition (ie. n = 1) there are only two possible arrangements. The initial sequence is represented as $S = S_1S_0 = 10$, and a permutation operation generates the output $P(S) = S_0S_1 = 01$. The following shows two groups of results:

Mate-state	S	1	0	P(S)	0	1
Function	J			P(J)		
0	0	0	0	0	0	0
$\bar{x}$	1	0	1	2	1	0
x	2	1	0	1	0	1
1	3	1	1	3	1	1

Figure 5. *Functions*

For any permutation operation, the function T(J) = P(T(J)) is always invariant. The inequality J ≠ K = P(J) holds in general.

Configuration space

Building upon the three spaces (variables, states, and functions), an additional space of organization is composed of permutation operations provided by permutation invariance properties defined in the previous section.

Complementary operation

Definition 5.1 (Complementary Operator) For any binary (0-1) variable y ∈ B2, let the relevant index $\delta \in B_2$ be a complementary operator:

$$y^\delta = \begin{cases} \neg \bar{y}, \delta = 0 \\ y, \delta = 1 \end{cases} \quad (7)$$

Definition 5.2 (Complementary Function Operation) For any n variable function of 2^n meta function vectors $S = S_{2^n-1}...S_I...S_0$ Let $\Delta = \delta_{2^n-1}...\delta_I...\delta_0$, $0 \le I < 2^n$, $\delta_I \in_2$, $\Delta \in B_2^{2^n}$.

For this type of complementary operations on function, Δ is :

$\Delta : T(J) \rightarrow K;\ J, K \in B_2^{2^n},\ 0 < J, K < 2^{2^n}$

$S\Delta = S_{2^{\wedge}n-1}^{\delta 2^{\wedge}n-1} \ldots S_I^{\delta I} \ldots S_0^{\delta 0},\ SI \in (B_2)^n$

$T(J)^{\Delta} = T(S_{2^{\wedge}n-1}^{\delta 2^{\wedge}n-1}(J_{2n-1})) \ldots T(S_I^{\delta I}(J_I)) \ldots T(S_0^{\delta 0}(J_0))$

$= T(J)_{2^{\wedge}n-1}^{\delta 2^{\wedge}n-1} \ldots T(J)_I^{\delta I} \ldots T(J)_0^{\delta 0}$

$= K_{2n-1} \ldots K_I \ldots K_0 = K \in B_2^{2^n}$

$T(J)_I^{\delta I} = T(S_I^{\delta I}(J_I)) = J_I^{\delta I} = K_I \in B_2$

$0 \le I < 2^n,\ 0 \le J, K < 2^{2^n},\ \delta_I \in \Delta$

Joined both permutation operations and complementary operations, a new level of configuration space is composed of a total of a total of $2^n!\times 2^{2n}$ elements elements [10,11].

Result and comparsion

Using this mechanism, different PRNG sequences can be generated and compared with a list of standard test methods.

Comparison test

The all algorithms were implemented by C language. This paper used the program to generate the pseudo-random binary sequence $1000*1000=10^6$, and then the NIST Statistical Test Suit test the binary sequence, the results are compared as follows.

In order to test VPRNG generates a random number quality, particularly selected with BBS, ANSI X9.17 and DES models to generate the same amount of random number (10^6), also use the same platform to test, the following results are recorded in Table 1.

Table 1. *Comparison test result*

Items	Comparison of content				
	VPRNG	X9.17	DES	BBS	VPRNG's rank
frequency	1.0000	0.9910	0.9880	0.9890	1
block-frequency	1.0000	0.9890	0.9880	0.9860	1
cumulative-sums	1.0000	0.9930	0.9870	0.9910	1
runs	0.9987	0.9920	0.9950	0.9910	1
longest-run	1.0000	1.0000	1.0000	1.0000	1
fft	1.0000	1.0000	1.0000	1.0000	1
nonperiodic-templates	1.0000	0.9760	0.9740	0.9760	1
overlapping-templates	1.0000	1.0000	1.0000	1.0000	1
universal	1.0000	1.0000	1.0000	1.0000	1
apen	1.0000	1.0000	1.0000	1.0000	1
serial	0.9893	0.9930	0.9850	0.9890	2
linear-complexity	0.9380	0.9200	0.9060	0.9330	1

By contrast, from the list of compared results, VPRNG on most items are equal or better than the other models.

Conclusion

It is important to design the VPRNG method to use variant logic construction. Since a pair of P and Δ operators and potentially have a huge configuration space $2^n!\times 2^{2n}$ times larger than classical Logic function spaces. If the Cell serial is defined by N, the Cell has a space of 2^N.

In upper example, N=1000 and n=3, so the total space is $2^{1000}*8!*256$. Exploring how difficulties for this mechanism to be decoded will be an important issue for coming cryptologist's theoretical targets.

Considering PRNG placed in the central position of stream cipher mechanism, and stream cipher technologies are more and more important in advanced network security environment, higher performance methodology and relevant implementation will be useful in future exploration.

Acknowledgement: Thanks to The Yunnan Advanced Overseas Scholar Funding and The School of Software, Yunnan University for financial supports to Information Security research projects.

References:

- [1] G.B. Agnew, Random source for cryptographic systems, *Advanced in Cryptology – EUROCRYPT'87 Proceedings*, Springer-Verlag, 1988, 77-81
- [2] D. Brumley and D. Boneh, Remote timing attacks are practical, *USENIX Security Symposium*, 2003
- [3] D. Eastlake, S.D. Crocker and J.I. Schiller, Randomness requirements for security, *RFC 1750*, Internet Engineering Task Force, 1994
- [4] M. Gude, Concept for a high-performance random number generator based on physical random noise, *Frequenz*, V.39, 1985, 187-190
- [5] J. Kelsey, B. Schneier, D. Wagner and C. Hall, Cryptanalytic attacks on Pseudorandom Number Generators, *Fast software encryption, Fifth International Workshop Proceedings*, Springer-Verlag, 1998, 168-188
- [6] Paul Kocher, Timing attacks on Implementations of Differ-Hellman, RSA, DSS and Other Systems, *CRYPTO 1996*, 104-113
- [7] Steven Murdoch, Hot or Not: Revealing Hidden Services by their Clock Skew, *ACM CCS* 2006
- [8] C. Plumb, Truly random numbers, *Dr. Dobbs Journal*, 19(13), 1994, 113-115
- [9] M. Santha and U.V. Vazirani, Generating Quasi-Random Sequences from slightly random sources, *Journal of Computer and System Sciences*, V.33, 1986, 75-87
- [10] Jeffrey Zheng and Chris Zheng (2010). A framework to express variant and invariant functional spaces for binary logic, *Frontiers of Electrical and Electronic Engineering in China*, Higher Education Press and Springer, 5(2): 163-172. *http://www.springerlink.com/content/91474403127n446u/*
- [11] Jeffrey Zheng, Chris Zheng, T L. Kunii (2011). A Framework of Variant-Logic Construction for Cellular Automata, *Cellular Automata-Innovative Modelling for Science and Engineering*. InTech Press 325-352 *http://www.intechopen.com/articles/show/title/a-framework-of-variant-logic-construction-for-cellular-automata*
- [12] Jeffrey Zheng (2011). Novel Pseudo-Random Number Generation Using Variant Logic Framework. *2nd International Cyber Resilience Conference*, 100-104 *http://igneous.scis.ecu.edu.au/proceedings/2011/icr/zheng.pdf*

MR. WEIZHONG YANG,

obtained a bachelor degree and a master degree in Yunnan University (2001, 2004). From 2004, Mr. Yang works in Department of Information Security, School of Software, Yunnan University as a Lecturer. Mr. Yang is interested in the information technology, especially focus on cryptography. Recently, Mr. Yang puts his attention to the variant logic and its applications on the Pseudo-Random Number Generation and its further applications on the stream cipher mechanisms.

DR. JEFFREY ZHENG,

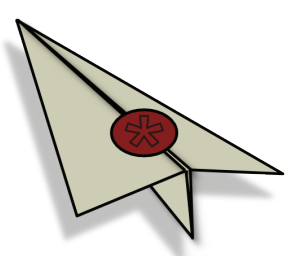
received ME and PhD degrees from USTC (1981) and Monash Uni. (1994) respectively. Dr. Zheng worked in Institute of Computing Technology, Academia Sinica as an assistant research professor in 1981-1987; Institute of Systems Sciences, NUS as a visiting scientist in 1987-1990; Victoria University of Technology as a research fellow in 1994-1997; CSRIO Division of Manufacturing Science & Technology as a senior software engineering in 1998-2002 and from 2004, Dr. Zheng is the director of Department of Information Security, School of Software, Yunnan University as a Professor. Dr. Zheng worked in parallel architecture & algorithm, image analysis and processing, content--based image retrieval, knowledge representation and higher performance of hierarchical modeling. In recent years, Dr. Zheng has focused his attention on variant logic construction and relevant applications on stream cipher mechanisms and advanced cryptology methodologies.

SAFESLINGER:
EASY-TO-USE AND SECURE PUBLIC-KEY EXCHANGE

MICHAEL W. FARB, YUE-HSUN LIN, ADRIAN PERRIG, JONATHAN MCCUNE

For many current Internet applications, users experience a crisis of confidence. Is the email or message we received from the claimed individual or was it sent by an impostor? Cryptography alone cannot address this problem. We have many useful protocols such as SSL or PGP for entities that already share authentic key material, but the root of the problem still remains: how do we obtain the authentic public key from the intended resource or individual? The global certification process for SSL is not without drawbacks and weaknesses, and the usability challenges of decentralized mechanisms such as PGP are well-known.

Of course, ordinary users can extensively rely on system administrators' help in making trust decisions. However, ordinary users inevitably face challenging decisions alone; most users at home, on travel, on vacation, or in small businesses do not benefit from skilled help. All this while the need and temptation to use new online services steadily increases.



The recent proliferation of smartphones offers a promising opportunity to address these challenges, as these devices offer a general computing environment with a powerful processor, high-resolution display, several communication modalities (Bluetooth, WiFi, 3G/4G), camera, and sensors.

Unfortunately, smartphone platforms suffer from many risks. Vulnerabilities exist in communication standards that enable eavesdropping or impersonation. Moreover, phone operators are disclosing information or introduce vulnerabilities through insecure or misconfigured systems.

Individuals often have physical interactions with resources or other individuals before communicating digitally. Often, people communicate over the Internet or via SMS after having met in person. We leverage this physical encounter to bootstrap digital trust.

Solution Roadmap

SafeSlinger is a system for secure exchange of authentic information between two smartphones, and a user interface for secure messaging. In essence, SafeSlinger exchanges contact information, containing public keys in addition to standard contact list information such as name, picture, phone numbers, email addresses, etc. Thanks to the association between the individual holding the phone and the public key that is exchanged, users (with the help of the SafeSlinger App) can later associate digital communication with the previously met individual by verifying a digital signature. To make SafeSlinger usable, the cryptographic aspects are mostly hidden from the user, and we have built-in several approaches to make SafeSlinger tolerant to user error.

We envision SafeSlinger as a general approach to bootstrap secure digital communication. (1) First, we've enabled groups (2-9 individuals) of physically co-located users to securely bootstrap trust by sending keys (**slinging keys**) between their devices (a one-time operation). SafeSlinger can also support remote setup, as long as users can authenticate the other individual (e.g., via telephone communication or live video conference). (2) Second, SafeSlinger supports secure phone-to-phone messaging and file transfer, providing both secrecy and authenticity. Once users' devices hold each others' public keys, the SafeSlinger user experience is nearly identical to that of traditional SMS and MMS messaging today. (3) Third, we will enable secure introductions without physical meetings by allowing a common acquaintance to facilitate a mutual introduction enabled by SafeSlinger's file transfer. (4) Fourth, we plan to release our source to enable other applications to adopt the SafeSlinger API to add their public key to a contact entry. Now, when a user sends (**slings**) its updated contact

list entry to another user, each application's public key is automatically included, and the same application at the other end can extract the public key. This mechanism can enable applications such as secure email or secure SMS to solve the problem of securely exchanging the public key without a leap of faith.

We have implemented SafeSlinger on Android and iOS, and it can be installed from their respective app stores to enable secure credential exchange. The public apps include mechanisms for secure messaging and file exchange for Android, with plans to release the same functionality for iOS in Summer 2012. Beyond that, we will extend SafeSlinger to provide secure introductions between two individuals.

Attacks to Resist

The main purpose of SafeSlinger is to enable a set of users to exchange their contact information such that every non-malicious user receives the correct information about every other non-malicious user. Malicious users may collude and impersonate each other, for example, therefore we cannot provide any guarantees for those parties. Our main goal is to provide high usability while preventing the attacks described later.

Secure local exchange of information is a surprisingly intricate and challenging problem. Possible attacks include:

- *Malicious bystander who participates in protocol:* a bystander can overhear conversation, and attack the protocol by controlling the local wireless communication. (Dolev-Yao attacker model). The Man-in-the-Middle (MitM) attack is a specific instance of this attack.
- *Malicious group member:* an invited member of the group wants to violate protocol properties, such as mounting an impersonation attack by injecting incorrect information for another user, or performing a Sybil attack by injecting multiple entries either for fictitious individuals or for individuals who are not present. A malicious group member can also perform a Group-in-the-Middle (GitM) attack, as described above.
- *Malicious server:* for protocols that rely on a back-end server, the server may be controlled by a malicious administrator or become compromised.
- *Information leakage after protocol abort:* an adversary may be able to cause a protocol abort and trigger leakage of private information about a participant.
- *Collision attack on low-entropy hash:* as described above, low-entropy hash values can be vulnerable to efficient birthday attacks if precautions are not taken.

Challenges to Overcome

We have also considered the following challenges, some of which directly apply to SafeSlinger, and others which we sidestep via the design choices that we make for SafeSlinger:

- *Exclusion of unintended participants:* legitimate users will expel an unwanted bystander who wants to participate in the protocol.
- *Correct member count:* users need to correctly count the number of group members who participate in an exchange.
- *Identity validation:* users correctly validate the identity information received from the exchange. They should map the identity information to the people who participate in the exchange, and they should reject information of non-participants.
- *Impersonation detection:* users verify that no other user has injected information that impersonates them in the current exchange. For example, a malicious user may also inject information about Alice, even though Alice is also participating in the exchange. The risk is that another user may discard the correct information and accept the wrong information.
- *Diligent hash comparison:* users can correctly perform a hash comparison, even after executing the protocol numerous times without any attack.
- *Diligent error checking and aborting:* users will abort the protocol and restart the protocol when suspicious or error conditions are encountered.

User Experience

Although the protocol is complex, the user experience is actually quite simple as SafeSlinger performs all the cryptographic operations and checks without the users' involvement.

The user experiences the following steps: (1) select the data items to be shared, (2) count and select the number of users, (3) find and enter the lowest ID displayed by the devices, (4) compare the word phrases and select the one that matches and click "match", or click on "no match", (5) select which users' data to import into its contact list.

Protocol Steps

In a nutshell, the mobile devices send their information to the server, which redistributes it to the other devices. The users then engage in a verification of all exchanged information to ensure that they all have received identical information. This verification is finished by the users who perform a comparison of textual representation derived from short hash values displayed by the phones.

Two problems must be avoided here: (1) users may habituate to click "OK" without performing the comparison, and (2) an attacker may compute a collision attack on the short hash value.

We solve (1) by presenting 2 decoy hash values alongside the correct value and asking users to verify which of the 3 hash values matches a value on other people's devices. We address problem (2) by using Short Authentication Strings (SAS), where, all devices first commit to the values that are used in the hash comparison. Once all the commitments are distributed, the devices reveal the decommitments and the short hash comparison can proceed. This approach prevents the collision attack and in Zimmermann's words converts the attack from a "safe attack" into a "daring attack."

Another challenge that we address in SafeSlinger is to prevent the server from learning any contact information. We accomplish this by leveraging a *group Diffie-Hellman protocol*. The group DH protocol establishes a shared secret key among all participants, which is used to encrypt the contact information. To prevent MitM attacks, the DH public key is included in the initial commitment and thus validated during the hash comparison.

In the following we go through the SafeSlinger protocol within each steps:

- *Data Selection & Counting:* the user selects which data to share and enters the total number of protocol participants.
- *Commitment, Group DH Key Setup:* each device computes the values needed for the group DH protocol by selecting the DH private key n_i at random. The device also randomly selects nonces to indicate "match" (Nonce match Nm_i) and

"wrong" (Nonce wrong Nw_i). The device also encrypts the data D_i to share with Nm_i used as a symmetric encryption key (using AES with 128-bit keys): $E_i = \{D_i\}_{Nmi}$. We also use SHA-1 truncated to 128 bits as hash function H. Figure 1 depicts this multi-value commitment structure for user U_i. Finally, the device sends the commitment C_i to the server.

- *Server Unique ID Assignment, User Grouping:* In the next step, the server groups the users. First, the server sends a unique ID to the device which the device displays and prompts the user to find the **lowest ID** amongst all devices. Then all users enter the lowest ID to their device and send it to the server.
- *Collection and Distribution of Initial Decommitment:* The server now knows which devices belong to the same group, and distributes ID and commitment pairs (ID_i, C_i) to all group members. Once a device receives all commitments, it opens up the first level decommitment (HN_i, G_i, E_i) (See Figure 1). If validation of all decommitments is correct, devices compute a hash over all decommitments of C_i, i.e., over the triplets (HN_*, G_*, E_*), sorted by the value of the unique ID_i assigned to each device to ensure that all devices compute the hash over the same triplet ordering.
- *Wordlist-based Comparison of Integrity of Commitments:* Each device then computes a word phrase that represents the hash -- we use the PGP word phrase which results in a representation that encodes 24 bits of the hash. If no phrase matches, the user selects "no match" and sends the "wrong" nonce Nw_i along with Hm'_i to enable verification to the server. This case is also triggered if the user selects the wrong word phrase. This approach cryptographically authenticates the "no match" message from the commitment C_i and thus prevents injection of the wrong nonce

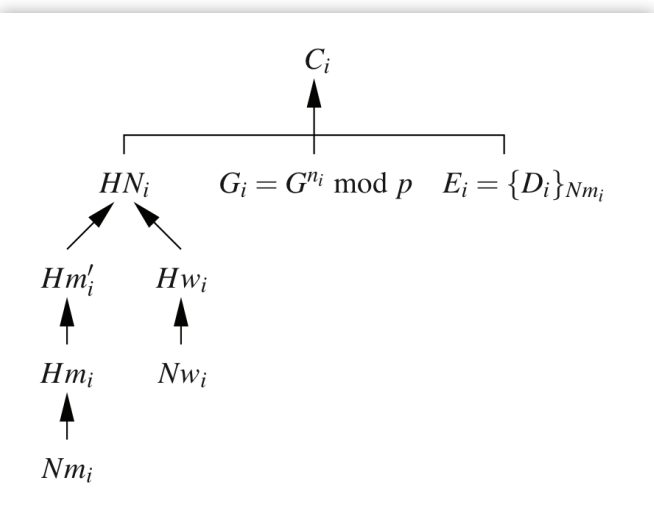


Figure 1. *Multi-value commitment structure for User U_i. C_i, HN_i, Hm'_i, Hw_i, Hm_i, Nm_i, are 160-bit values; G_i is 512 bits, and Di varies in length. Each arrow imples SHA-1 hash operations.*

by an adversary. Later, users correctly selected the matching word phrase, and the device reveals the pair of values indicating success (Hm_i, Hw_i), which the server redistributes to other devices.

- *Group DH Key Establishment:* Each device can verify that all the users selected the correct word phrase and the devices proceed to construct the group DH tree. The ordering in the tree is determined by the sorted order of the unique ID_s ID_*. Since the group DH protocol we use is intricate, we omit the details for enhanced readability.
- *Distribution and Verification of Data Decryption Key:* Once the secret group key K is established, the devices then proceed to send their final match nonce Nm_i (which serves as the data decryption key) to the group, encrypted with K.
- *Decryption of Data and User Acquisition:* In the end, each device decrypts and verifies the correctness of Nm_i, and finally uses Nm_i to decrypt the data D_i.

Architectural Features

SafeSlinger provides several convenient and useful features based on its design.

- *Fast, Consistent, Wireless Communication:* We currently target Android and iPhone devices, with an effort to make design decisions compatible with future implementations for other platforms (e.g., Windows Phones). Unfortunately, today's platforms do not offer consistent support for 802.11 ad-hoc mode or seamless creation of a base station to enable other devices to connect to them. Bluetooth communication is also inconvenient because of the slow discovery phase and the inability of iPhones to communicate with non-Apple devices (excepting headsets). NFC is not yet widely deployed, and such communication does not scale beyond pairwise communication. As a consequence, we use Internet-based communication, where all the mobile devices connect to a cloud server. This approach has the additional advantage that no latency for device discovery is experienced, as the devices can simply send packets to the server via IP.
- *Scale:* Though the current technical limit for our protocols and implementation is much greater than 9 users, it is unclear that there is much value in scaling even this far due to human limitations. We concentrate our presentation on groups of up to 9 users, leaving it as an open question whether there is a need for protocols that scale further. As SafeSlinger protocol fails to complete if only a single person miscounts, we set the threshold at 9 users. Asking users to count the number of participants rules out several attacks.
- *Grouping:* When mobile devices initially connect to the server, the server does not know which devices belong to the same group. It is a challenging problem for the server to determine the grouping, especially if several concurrent exchanges are ongoing. We employ the following approach that does not leak any sensitive information to the untrusted cloud server. The server assigns a unique ID to each mobile device, which it displays to its user. The devices prompt the users to find and enter the lowest ID. The devices then send that ID back to the server, which can thus perform the grouping. Note that the actual grouping is not security-sensitive, as an intruder can only cause denial of service.
- *Confidentiality During Data Exchange:* All exchanged data is encrypted and the actual encryption key K_D and initialization vector IV_D are derived from the 160-bit "match" nonce Nm_i. Contact data integrity is achieved through verification of the commitment C_i, hence, no additional Message Authentication Code (MAC) is needed. Since we can validate Nm_i based on the commitment Hm_i, no additional MAC value is needed to ensure integrity and authenticity for that encryption.
- *Word Phrase Verification:* In SafeSlinger, the word phrase is constructed from the first 24 bits of the 160 bit SHA-1

hash. We use the standard PGP approach for converting a 24-bit value into 3 words. PGP uses two word lists -- an "even" and "odd" list -- with 256 words each. Based on the standard PGP approach, the first 8 bits select a word in the "even" list, the second 8 bits select a word in the "odd" list, and the final 8 bits select another word from the "even" list. We discourage careless comparison by displaying two unique (across all devices in the exchange) decoy phrases in addition to the common phrase.

- *Word Phrase Collision Avoidance:* Although unlikely, the words in a decoy phrase may match the words in a decoy phrase on another device, causing the user to select the decoy phrase which results in an error detected by the local device. We want to avoid true randomness in the decoy phrases so that careless users will not chose the wrong phrase if the actual hash phrase and either of the decoy phrases contain the same word in the same position. We thus chose our decoy phrases deterministically such that each decoy word will be unique across all decoy phrases displayed in the group.
- *Address Book Key Management:* We rely heavily on the mobile operating systems' contact list facilities to manage users' contact data and public keys. It is convenient to store users' public key data in a recognizable field in the smartphone's address book, so that any existing synchronization service will seamlessly maintain backups. We realize this functionality by adding the name and value (base-64 encoded public key) of a new instant messaging (IM) provider to the contact list.

Applied Key Exchange

We have implemented SafeSlinger as a base API library on both Android and Apple iOS platforms. In the future, any third-party application for either platform may link against or execute the key exchange (with its GUI).

Cryptographic operations are computed using the operating system-provided libraries for Android and iOS, with the addition of open source OpenSSL libraries for Apple iOS. Figure 2 shows the information flow between multiple devices during execution of the key exchange, outlined as follows.

- 3rd-party app generates a public/private key pair.
- 3rd-party app inserts its public key in the device's contact list.
- 3rd-party app invokes the key exchange API, specifying the appropriate contact list entries (with the name of the public key(s) to exchange).
- During the SafeSlinger key exchange protocol, multiple messages are exchanged between devices via our server, and validated independently by each device.
- The newly received (and authenticated) public key(s) and contact data are saved in the device's contact list.
- 3rd-party applications may now make use of newly imported public keys from the contact list.

Applications

We have implemented secure rich messaging for Android and iOS. Secure information and shared public keys via Key Exchange are used to encrypt and authenticate text messages and file data. When SafeSlinger has been installed, it generates an RSA 2048-bit key pair first. The application then obtains a Google Android C2DM Push token or an Urban Airship token for addressing the Android or iOS devices.

During a SafeSlinger information exchange, the push tokens of all group members are exchanged and imported into the address book alongside the public keys.

Some interesting potential uses for SafeSlinger are:

- Secure Text Messages: Separate corporations which already manage internal employees, each with a large PKI, who want to share sensitive materials, do not have to choose one internal PKI to use, as SafeSlinger Messaging can bridge the gap between large but separate existing PKIs.
- Secure File Transfer: Hospitals that want to remotely share test results or imaging data with their doctors or patients, can do so confidentially through a SafeSlinger secure file exchange on the patient's phone.

Secure Introductions

A future implementation could leverage our secure file transfer mechanism to enable secure introductions. A common friend of two users sends contact data that includes public keys, to each other. More concretely, consider Alice with two friends: Bob and Carol. Alice has performed a SafeSlinger exchange with both Bob on one occasion, and with Carol on another, and has thus received an authentic SafeSlinger public keys for both Bob and Carol. Likewise, both Bob and Carol have Alice's authentic SafeSlinger public key. In a secure introduction, Alice first encodes Bob's contact information (which includes Bob's SafeSlinger public key and Push token) as a custom vCard and uses an OpenPGP message format to provide secrecy and authenticity. Alice then sends this message via SafeSlinger to Bob. Hence, Bob can validate that the information indeed originates from Alice, whom he trusts not to send bogus information. Analo-

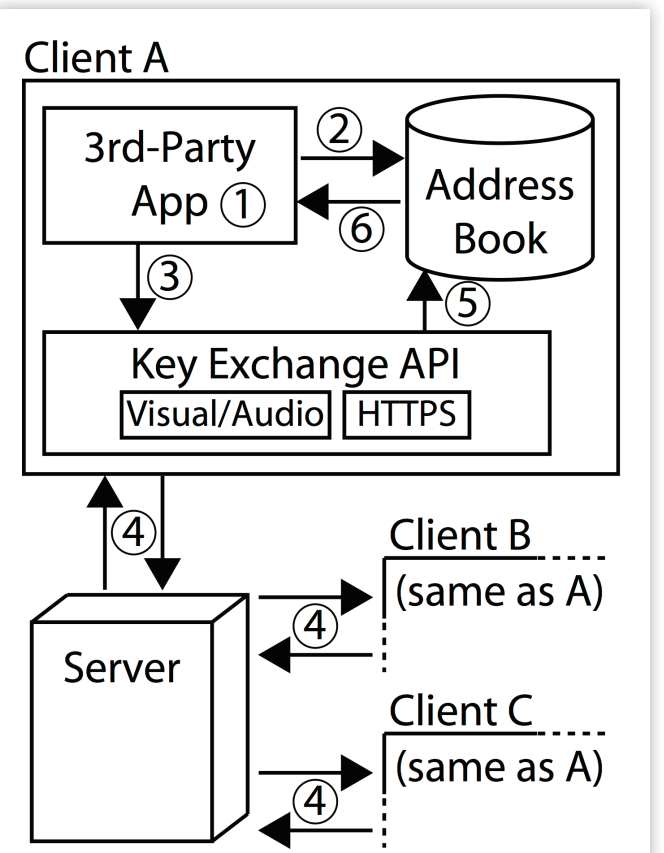


Figure 2. *SafeSlinger keyexchange API interactions with 3rd-party applications.*

gously, Carols trusts Bob's information received from Alice. Now that Bob and Carol have each other's public keys and push tokens, they can use SafeSlinger to securely communicate.

Summary

To realize the vision of secure online communication, we need to overcome several human challenges: some users are ambivalent about security or privacy, most users lack security expertise, and many users prefer convenience over security and may not want to expend much effort for security. To counteract these challenges, we designed SafeSlinger as an easy-to-use application that offers many benefits to drive usage.

We have released our SafeSlinger application both for Android and iOS devices with the intention to provide a free and easy-to-use system that enables secure communication. Through free multi-platform applications available on smartphone markets, open documentation, and open-source code, we anticipate wide adoption of SafeSlinger. Assuming wide adoption, we hope to provide usable and secure communication for the masses, and a security platform that will enable numerous security services and applications. A more detailed technical white paper[1] can be found at our website[2], and our applications can be installed from the Google Play Store and iTunes App Store.

MICHAEL W. FARB

joined Carnegie Mellon CyLab as a Research Programmer in 2010. He received his BA from Beloit College in 1995, and as a mobile device software developer, has worked in publishing, transportation, and security.

YUE-HSUN LIN

joined Carnegie Mellon Cylab as a Postdoctoral Researcher in 2012. He received his PhD from National Tsing Hua University in 2010. His research includes wireless security, sensor network security, and secure protocols design.

ADRIAN PERRIG

is a Professor in Electrical and Computer Engineering, Engineering and Public Policy, and Computer Science at Carnegie Mellon University. Adrian serves as the technical director for Carnegie Mellon's CyLab. He earned his Ph.D. degree in Computer Science from Carnegie Mellon University.

JONATHAN MCCUNE

is a Research Systems Scientist for CyLab at Carnegie Mellon University. He earned his Ph.D. degree in Electrical and Computer Engineering from Carnegie Mellon University, and received the A.G. Jordan thesis award. He received his B.Sc. degree in Computer Engineering from the University of Virginia (UVA).

[1] *http://sparrow.ece.cmu.edu/group/pub/SafeSlinger.pdf*
[2] *http://www.cylab.cmu.edu/safeslinger*

If our FREE antivirus for home outperforms competitors' end-point products, imagine what our business solutions can do for you.

avast!
be free

The most popular antivirus in the world.
www.avast.com/best-antivirus

OVERVIEW OF SIDE CHANNEL AND TIMING ATTACKS

MARTIN RUBLIK

Attacking a system is a tradeoff between the attacker's possibilities and gains and be-tween the difficulty of an attack. The article will focus on special techniques that attacker can use to gain valuable information only by observing the system. Though these types of attack are not easy to execute, the attacker can gain a lot. These attacks can be used to subvert a system that is secure under a commonly accepted threat model. A system can be secure in theory, but the at-tacker can subvert the system's defense by exploiting the vulnerabilities that were either not included in the threat model or emerged during the implementation phase.

Introduction

Based on what the attacker knows about the target we can divide the attacks into several categories:

- attacks where the attacker does not know anything about the target,
- attacks where the design of the target system is known,
- attacks where implementation of the target system is known or
- attacks where details about the implementation and environment specific details are known (such as software and hardware equipment, services topology, etc.).

Some systems rely on obscurity as a defense mechanism and this is where the first class of attacks belongs. In general this is not a good tactic, especially in cryptography. Modern cryptographic systems are designed with a premise that the internals of the system are known to the attacker. The reason is that it is really hard to keep secret about the design of a system, as attacker can usually reverse engineer the system easily. Several examples support this design principle. Among the others the most known failure was design of DVD CSS algorithm used for DVD protection. This algorithm was broken by Jon Lech Johansen who was only 16 year old at the time.

Attacking the system design is mostly a theoretic task, but breaking it, has severe consequences to the system and its practical use. In cryptography these types of attack are mostly algorithmic attacks and are of course implementation independent. Therefore when a *practical* algorithmic attack on cryptographic system is found the system needs to be replaced where applicable. An example of such an attack would be design flaws in WEP [1] that lead to WPA/WPA2 rollout or flaws found in MD5 hash algorithm [2] that lead to global hash algorithm change in X.509 certificates.

The third class of attacks is implementation specific and in general can be "easily" patched. However the patching possibilities may vary depending on the system's characteristics. For example it is easier to patch a network connected server operating system vulnerability, but it is much harder to patch an offline embedded system.

An example of implementation specific flaw would be the infamous Debian cryptographic random number generator flaw that allowed the attacker to brute-force a private cryptographic key generated on a Debian server [3] easily. The issue was that the key-space used for generating the private keys was reduced less than 300K keys due the bug in the random number generator. PRNG was incorrectly seeded by process IDs ranging from 0 to 32767 providing not enough seed entropy (only 15 bit) for generating the private keys.

Finally attacks, where special hardware or software equipment is known to the attacker are applicable only in special situations. On the other hand attacker has most knowledge about the targeted system, thus he can exploit system in several ways. An example of implementation and environment dependent attack on a system is a side channel attack.

Side channel attacks

Side channel attack is an attack where attacker infers valuable information by observing the system's behavior. In general side channel attack analyses leaked information with regards to hardware, software, implementation and design specifics of the targeted system.

Side channel attacks can be either passive or active. In a passive side channel attack the targeted system leaks information within the ordinary communication and the attacker deduces the results only by observing the system. In an active side channel attack the attacker can actively manipulate the

system and its hardware in either an ordinary or invasive / malicious way so it leaks useful information.

Active side channel attacks can exploit faults of a system (fault attacks). In these cases the attacker brings system to erroneous states in order to observe errors. An example of fault attack is presented in [4] and involves stressing a smart-card with high voltage or changing the system clock signal. These actions can lead to revealing information about stored secret keys protected by the smart-card in some cases.

A system can leak information by various means. Most known side channels include:

power analysis electromagnetic emissions (EMM) acoustic analysis	Attack requires close access to the target in order to gather information.
traffic analysis timing attacks	System observation *can* be performed remotely. It is possible to mount attacks against remote targets.

The power analysis is in general a non-invasive attack. Attacker measures the power consumption during the computations done by a system which requires physical access to the target. This type of attack is therefore executed mostly against tamper evident / resistant hardware such as smart-cards and hardware cryptographic modules in order to gather information about the protected cryptographic keys. For systems running on common hardware there are easier ways to subvert the system using physical access, such as installing malicious software or utilizing hardware key-loggers. The idea behind the power analysis is that the targeted system consumes different amounts of power depending on the bit structure of a key. For example it is possible to reveal the private key of *unprotected* RSA implementation that uses square-and-multiply algorithm for modular arithmetic. In a square-and-multiply algorithm the power consumption varies significantly [5] when the private exponent's current bit is 0 or 1. Therefore by analyzing the consumption during the computation with a private key, the attacker can deduce the private exponent of RSA key pair. Practical example of power analysis was recently presented in research done by Sergei Skorobogatov (*University of Cambridge*) and Christopher Woods (*Quo Vadis Labs*). Using power analysis these researchers found a backdoor in a commercial field programmable gate array used also in a military sector [6].

Similar to power analysis is an attack technique based on measuring the electromagnetic emissions. In order to measure the electromagnetic emissions it is necessary to have close access to the system, but the attacker does not need the physical control over the system. EMM attacks can be executed against cryptographic tamper evident / resistant systems in manner similar to power analysis attacks, but can be useful for compromising a general system as well, because the physical control over the system is not strictly required. Van Eck (a Dutch computer researcher) described a process that used the EMM analysis in order to eavesdrop on the contents of CRT monitor [7]. This type of attack was later extended to eavesdropping on LCD panels as well. Another famous EMM analysis was performed by Martin Vuagnoux and Sylvain Pasini (*Security and Cryptography Laboratory/ EPFL*). They took advantage of electromagnetic emissions in order to eavesdrop on wired and wireless keyboards [8].

Finally the third class of side channel, that requires close access to the targeted system in order to observe the leaked information, is acoustic analysis. During acoustic analysis the attacker monitors sounds emitted by components of the system. Intuitive examples of such components would be keyboards [9] or printers [10]. Less intuitive example is exploiting sounds emitted by CPU. Different operations performed by a processor have different sound signatures. Based on this information it is possible to identify what type of operation is performed by CPU. Attacker can misuse this information in a combination with different type of side channel attack, for example in a timing attack [11].

Side channel attacks where system can be observed remotely are more practical for breaking into common systems for an obvious reason: the attacker does not need physical access to the targeted system (which is in general hard to gain). One of the side channel attacks that could be mounted remotely is traffic analysis. Traffic analysis is not a new concept and was used in military circles long before computer security. In computer security traffic analysis can be used to gain information based on size, *frequency* and *timing* of network communications. Traffic analysis is therefore in a close relationship with timing attacks. A practical example of traffic analysis was recently provided by *IOActive Research Labs*. In their proof-of-concept [12] they showed that it is possible to gather information about the places visited through *Google Maps™* over SSL encrypted connection by analyzing the encrypted network communications.

Last class of side channel attacks analyses system's response time in order to gather valuable information. This type of attack is called timing attack and we will describe it further in the article.

Timing attacks

Side channel timing attacks are commonly used in order to gather information that is not very large, does not change during the attack (at least not in a significant way) and is used as computation parameter during the attack. An example of such information would be cryptographic keys or passwords.

Timing attack was first introduced in scientific literature by Paul C. Kocher [13] in 1996. This paper is cited in almost every scientific article concerning timing attacks and in the paper Paul Kocher describes an attack that can recover RSA private cryptographic keys. This attack is not computationally demanding and requires only a few thousand cipher-text samples, which made it implementable in practice at the time of publishing the results.

Since then the timing attacks got even more practical. First timing attack that was applicable against a widely implemented solution was timing analysis of keystrokes in SSH protocol [14]. This attack combined traffic analysis of SSH encrypted communications and timing analysis of inter-keystroke information of user's typing. The researchers performed a statistical study on users typing patterns that improved a brute-force password attack by factor of 50. Using a special algorithm for guessing the passwords the researchers reduced the password space to 1/50 in common case.

Later on a *remote* timing attack was presented by David Brumley and Dan Boneh (both from *Stanford University*) [15]. They attacked *Apache mod_SSL/OpenSSL* and were able to extract private keys over a local area network that included several network segments divided by routers and switches. This was a major breakthrough as until then, many people believed, that remote timing attacks were not practical. Until then most people also thought that timing attacks were applicable only to special hardware such as security tokens or smart-cards because decryption times would be masked by many concurrent processes running on a common operating system (such as web server). The attack took advantage of key negotiation in SSL protocol. In the SSL key negotiation phase the server decrypts data chosen randomly by the client in order to prepare a shared secret that is used for protecting the communication later. The attacking client sent malformed data, that made the server break communication with

an alert response. The time elapsed between sending the data and receiving the alert was used as observation sample by the attacker. The attack took about two hours and required around a million queries in order to extract 1024 bit RSA key.

Finally timing attacks were mounted in practice against commonly used consumer products. In 2007 a timing attack was used to crack *Xbox 360™* console [16]. The goal was to replace the Xbox operating system with an older one. The flaws in older Xbox kernel versions lead to possibility of running pirated software. The core of the attack resided in *memcmp* function that is used for comparing two memory blocks. This function was also used for comparing the HMAC values used to authenticate the boot-loader/kernel. The *memcmp* flaw resides in fact that the comparison is done byte-by-byte and will end immediately after the first difference is found. This flaw was exploited as follows. The attacker started the HMAC comparison with a random value and varied the first byte. He measured the response time for all comparisons (bytes 0, …, 255) and chose the value with the longest measured time. This way the attacker can guess the first byte of the HMAC based on the fact that the comparison with correctly guessed byte will take a longer time (as the other comparisons will terminate immediately). The time difference between correct and incorrect guess was about 2200 microseconds. By iterating the guess cycle the attacker finally he got all the necessary bytes of the correct HMAC.

Timing attacks can also be used in situations that do not strictly belong to side channel attacks. In these situations the attacker uses the timing information for creating a covert channel that transfers the information from a compromised system. An example of such covert channel could be illustrated by blind SQL injection.

In blind SQL injection the system is prone to SQL injection but the attacker is unable to download results of the exploit through the web page in a traditional way. However he can guess the results based on the response time of a server. Following listings illustrate the possible SQL statements that can be used in order to infer this type of information (Listing 1).

The first query makes use of *MySQL* BENCHMARK command. This command is used for multiple executions of an expression. By running high number of operations (such as generating a MD5 hash) we can create a time delay. If the user *root* is present in the table *users* then the query will run for a longer time, otherwise it will finish almost immediately. This way the attacker can determine whether *root* access is enabled on a *MySQL* server (Listing 2).

A similar technique is illustrated in listing 2, though the SQL code is targeted on MS SQL database engine. If the user that is accessing the database is a domain administrator the query execution lasts more than 10 seconds.

Finally the third listing illustrates an attack that makes use of a time demanding query. The command selects and counts the result over a large table created by a series of cross joins. A clean SQL server installation has about sixteen users in *sysusers* table. The number that should be the result of the query is therefore quite big ($16^8=2^{32}$) and the join / count takes a while. The query will run for a longer time only if the first (highlighted) part of the condition is true, otherwise the SQL optimizer will stop the query and return an empty set. This type of optimization is also known as early exit and represents the core idea of all timing attacks. The attacker starts with guessing the first character of the currently connected user and ends when he guesses the entire logon name. The attacker code could be optimized by performing a binary search on the current guessed character in the user name, instead of using simple substring comparison (Listing 3).

Though blind SQL injection attack is time consuming, it works every time the system is prone to SQL injection and it is easy to automate and execute. Attacker can even take advantage of several tools for automating SQL injection, including famous *sqlmap*.

The results of blind SQL injection attack or attacker introduced time related covert channel can be accurately distinguished, because the attacker controls the execution time. This is not true for classic side channel timing attacks. The classic timing attacks leak information with much smaller variance. In most cases it is necessary to collect many measurements in order to mount successful timing attacks.

Nate Lawson and Taylor Nelson gave a talk at Blackhat 2010 [17] that described methods and procedures for mounting a timing attacks in practice. In general the steps of executing a timing attack can be summarized as follows:

- Set up the client to query the target and measure typical response time
- Vary the parameters, take many samples per guess and prepare a statistical base

Listing 1. *MySQL query for checking if* ***root*** *user exists in user database*

```
      IF EXISTS (SELECT * FROM users WHERE username = 'root') BENCHMARK(100000000,
MD5('expression'))
```

Listing 2. *MSSQL query for checking whether current connected user is a domain administrator*

```
      IF (SELECT SYSTEM_USER)='DOMAIN\Administrator' WAITFOR DELAY '0:0:10'
```

Listing 3. *MSSQL (time demanding) query for checking whether current connected user name starts with DO*

```
      SELECT 'FOO'

      WHERE

      SUBSTRING(SYSTEM_USER,1,2)='DO' AND

      (SELECT count(*) FROM sysusers AS sys1, sysusers as sys2, sysusers as sys3,
sysusers as sys4, sysusers as sys5, sysusers as sys6, sysusers as sys7, sysusers
as sys8)>1
```

- Perform a statistical analysis, filter the base
- Guess the partial secret
- Repeat the steps until the entire secret is revealed

The most important steps are the statistical base preparation and the analysis. It is necessary to collect and filter the results so that the positive and negative guess can be distinguished. An example of such basis is illustrated by following figure.

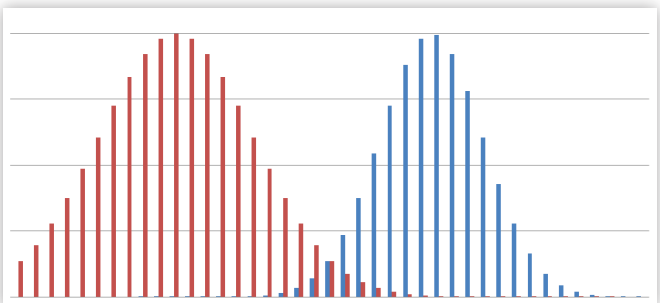


Figure 1. *Sample histogram of measured events*

The sample histogram illustrates two easily distinguishable sets of events (time measured when the test was run with first set of parameters is illustrated by red color the other one is blue). This way the attacker can decide with a reasonable amount of certainty which guess was correct. If the histogram is not that perfect the attacker could try to enlarge the base, use statistical filtering methods, vary the parameters or try to change the environmental circumstances so that he can take more accurate measurements. If the attacker is not able to distinguish the sets the system is not vulnerable to timing attacks.

Defense techniques against side channel timing attacks

The obvious defense against timing attack is to eliminate all the leaks, so that system finishes computation in constant time regardless of the input. This is especially important for security sensitive operations such as operations with cryptographic or security characteristics, but can be time intensive and could degrade the performance for other non-sensitive operations.

Another possibility is to introduce random delays / noise to the affected processes. This could look effective at first glance, but the attacker can bypass this protection by increasing the number of measurements [13]. In general the number of samples increases quadratically with the timing noise. This means that if the standard deviation between the samples was *10ms* and the added noise was normally distributed with 1 second standard deviation, then if the original attack required 1000 operations for successful outcome, the attacker will need to perform $(1000ms/10ms)^{2}*1000$ operations.

Another possible defense relies on detecting an attack. This may be possible depending on circumstances. For example it is suspicious if the same client sends invalid HMACs or malformed SSL HELLO requests to a web server. After several attempts the attacker should be blocked, thus rendering the attack unsuccessful. On the other hand there may be situations where the attack is not detectable. These attacks typically make use of common legitimate traffic (such as the SSH attack described earlier in the article) and blocking the traffic is therefore not an option.

Of course other defense techniques that depend on the nature of the attacked operations exist. One can use a replacement operation that is not vulnerable to timing attack or modify the operation in a way that makes the attack impractical. An example of such modification is RSA timing attack protection [13]. The protection resides in *blinding* (multiplying) the cipher-text by a random number before the exponentiation. This process prevents the attacker from knowing what bits are processed and makes bit-by-bit analysis impossible.

Conclusion

Easier ways than side channel attacks exists for subverting the system's security. A targeted attack on the system with specially crafted malicious PDF file or a phishing scam is much easier to prepare and quite likely to succeed. However the history shows us that side channel and timing attacks can be practical and security designers should include them in their threat modeling, especially when designing security sensitive applications and embedded systems that are going to be used for a longer period of time.

MARTIN RUBLÍK, PH.D., CISSP

Martin Rublík is a senior security consultant at BSP Consulting and a lecturer at University of Economics in Bratislava. He received his master's degree at Faculty of Mathematics, Physics and Informatics (Comenius University in Bratislava) in 2005 and his Ph.D. degree at Faculty of Economic Informatics (University of Economics in Bratislava) in 2010. His main area of expertise includes identity management, PKI, smart cards and network security.

References

- [1] Andrea Bittau, Mark Handley, and Joshua Lackey, „The Final Nail in WEP's Coffin – IEEE Symposium on Security and Privacy" Oakland, 2006.
- [2] Marc Stevens and Alexander Sotirov and Jacob Appelbaum and Arjen Lenstra and David Molnar and Dag Arne Osvik and Benne de Weger. (2009) Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. [Online]. *http://eprint.iacr.org/2009/111*
- [3] Luciano Bello. (2008, May) DSA-1571-1 openssl – predictable random number generator. [Online]. *http://www.debian.org/security/2008/dsa-1571*
- [4] Hamid Choukri, David Naccache, Michael Tunstall and Claire Whelan Hagai Bar-El. (2004, April) The Sorcerer's Apprentice Guide to Fault Attacks. [Online]. *http://eprint.iacr.org/2004/100*
- [5] Pankaj Rohatgi. (2010, May) Protecting FPGAs from power analysis. [Online]. *http://www.eetimes.com/design/programmable-logic/4199399/Protecting-FPGAs-from-power-analysis*
- [6] Wim Van Eck. (1985) Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk? [Online]. *http://cryptome.org/emr.pdf*
- [7] Rakesh Agrawal Dmitri Asonov. (2004) Keyboard Acoustic Emanations. [Online]. *http://rakesh.agrawal-family.com/papers/ssp04kba.pdf*
- [8] Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, Caroline Sporleder. Michael Backes. (2010, August) Acoustic Side-Channel Attacks on Printers. [Online]. *http://static.usenix.org/event/sec10/tech/full_papers/Backes.pdf*
- [9] A. Shamir E. Tromer. (2004) Acoustic cryptanalysis. [Online]. *http://www.cs.tau.ac.il/~tromer/acoustic/*
- [10] Sylvain Pasini Martin Vuagnoux. (2009) Compromising Electromagnetic Emanations of Wired andWireless Keyboards. [Online]. *http://www.usenix.org/events/sec09/tech/full_papers/sec09_attacks.pdf*
- [11] Christopher Woods Sergei Skorobogatov. (2012, March) Breakthrough silicon scanning discovers backdoor in military chip (DRAFT of 05 March 2012). [Online]. *http://www.cl.cam.ac.uk/~sps32/Silicon_scan_draft.pdf*
- [12] IOActive Research Labs. (2012, February) I can still see your actions on Google Maps over SSL. [Online]. *http://blog.ioactive.com/2012/02/ssl-traffic-analysis-on-google-maps.html*
- [13] Paul C. Kocher, „Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in Advances in Cryptology—CRYPTO'96, Santa Barbara, California, USA, 1996.
- [14] Dan Boneh David Brumley. (2003, August) Remote Timing Attacks are Practical. [Online]. *http://www.usenix.org/events/sec03/tech/brumley/brumley.pdf*
- [15] David Wagner, Xuqing Tian Dawn Xiaodong Song. (2001, August) Timing Analysis of Keystrokes and Timing Attacks on SSH. [Online]. *http://static.usenix.org/events/sec01/full_papers/song/song.pdf*
- [16] (2007, December) Xbox 360 Timing Attack. [Online]. *http://beta.ivc.no/wiki/index.php/Xbox_360_Timing_Attack*
- [17] Taylor Nelson Nate Lawson. (2010, September) Exploiting Timing Attacks in Widespread Systems. [Online]. *http://www.youtube.com/watch?v=idjDiBtu93Y*

THE DICHOTOMY OF SYMMETRIC VS ASYMMETRIC CRYPTOGRAPHY

WAYNE PATTERSON

Two roads diverged in a yellow wood,
And sorry I could not travel both …

Robert Frost, "The Road Not Taken", 1920.

In this article we will describe a fundamental dilemma in the world of cryptography, because of significant differences in the two types of cryptography in use today. The concept of symmetric cryptography is at least as old as Julius Caesar, and has been the only approach throughout history until the last generation. Asymmetric cryptography was only conceived in the 1970s, but it solves certain problems that cannot be addressed in the symmetric mode. Yet asymmetric cryptography introduces difficulties of its own. The challenge of resolving these two approaches at the algorithmic level is wide open.

Cryptology is an ancient field of study, with its origins going at least back to the era before Christ. However, whereas encryption for centuries dealt with transformations of symbols from natural languages, the invention of the computer introduced a more important symbol set as the basis for cryptology, the set {0, 1}.

Furthermore, with the use of binary symbols and the underlying environment for transforming messages becoming the computer and the networks to which computers were attached, new problems arose that eventually realized the famous quote from Robert Frost cited above. And so cryptology has faced its diverging paths, and one might say that the road has not yet been chosen, and may never do so.

In order to describe this dichotomy between two differing approaches that we will define as *symmetric* and *asymmetric* cryptography, it will be necessary to review the techniques developed over centuries and why in certain instances they have failed to provide solutions in the context of modern-day communications.

The first two millennia of symmetric cryptography

Historically, most encryption systems have been based either on the concept of substitution, or of transposition, or both.

Substitution

One of the earliest known cryptosystems is usually referred to as the *Caesar shift,* after Julius Caesar [1]. The technique Caesar shift uses is a simple substitution of the symbols used in communication. For simplification, let us suppose that we are encrypting a message (called the *plaintext*) that uses the 26 symbols from our Roman alphabet. In order to encrypt, we will write the letters in order, and then advance them by a fixed number of positions in the alphabet, understanding that the letter following Z will be A, and so on. Thus the letters of the alphabet in the encryption will be substituted by the corresponding letter shifted the chosen number of positions, leading to an encrypted message or *ciphertext*. For example, if the number of positions shifted is eight, the correspondence and hence the substitution will be the following:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H

Therefore, if we wish to encrypt the message "HAVE A NICE DAY, JULIUS", we would perform substitutions as indicated above so as to yield:

H	A	V	E	A	N	I	C	E	D	A	Y	J	U	L	I	U	S
P	I	D	M	I	V	Q	L	M	L	I	G	R	C	T	Q	C	A

Using this choice of eight positions in the Caesar shift implies that the sender and receiver of the message would only have to agree first, that they were using this encryption method, and second that the number of positions to be shifted was 8. This additional information (the number 8) to be communicated is called the key.

It is certain that this encryption method provides almost no security, since a person intercepting the message, should he or she assume that the technique being used is the Caesar shift, would only have to try all of the 26 possibilities for keys before the original message was uncovered. Thus, as a first principle of cryptology, we must in the design ensure that the number of possible keys is so large that is infeasible to simply try all the possibilities--a technique known as *exhaustive search* or *brute force*.

Another way of creating a substitution-based cryptosystem using the symbols of the Roman alphabet is to assume that the key will be any possible permutation of the 26 letters. We know from algebra that the total number of such permutations is 26!, which is:

403,291,461,126,605,635,584,000,000

At one try per second, exhaustive search would take more than a quadrillion centuries to perform. However, a weakness of this system is that it will be more difficult to communicate the key--that is, the specific permutation--since we could hardly recall this from memory and so it would have to be written down and passed somehow from sender to receiver.

This demonstrates another problem with the communication of a key or keys in a cryptosystem, called the *key management problem*. Prior to all appropriate parties having possession of the necessary key or keys, the channels of communication planned cannot be used. Hence, an alternative method to distribute keys must be utilized. In today's environment, if the communications channels are within a computer network, the alternative method, for example, might be the use of a separate courier system.

Over time, many different and ingenious encryption systems were devised using the principle of substitution. One was keyword substitution [1], where a word (easily remembered) was chosen and written under the original alphabet, and then the other letters in their order. For example, if the keyword was "FACETIOUSLY", then the specific substitution would be:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
F	A	C	E	T	I	O	U	S	L	Y	B	D	G	H	J	K	M	N	P	Q	R	V	W	X	Z

And the same message we used above would encrypt as:

H	A	V	E	A	N	I	C	E	D	A	Y	J	U	L	I	U	S
U	F	R	T	F	G	S	C	T	E	F	X	L	Q	B	S	Q	N

It should be noted for the benefit of codebreakers (or cryptanalysts) that the situation is not as bleak in breaking these substitution techniques as the use of exhaustive search. In fact, solvers of the newspaper puzzles often described as *cryptograms* have learned that a better technique to break this type of substitution is to use *letter frequency analysis*, that is, using the recognition that in most samples of English text, certain letters appear far more frequently than others. In most cases, with large enough samples of text, the letter "E" will occur as much as 50% more often than any other letter, with usually "T" next in order of frequency. Thus in a substitution, one counts which encrypted letter appears most often, and proceeds with the assumption that this is the symbol substituted for "E" and goes from there.

Transposition

Historically, a second approach used for encryption was that of transposition. Rather than replacing one symbol by another, one shifts the location of the symbol according to some permutation of the positions where the symbol is located in the message. For example, if we are to assume there are 10 letters in the message, and the chosen permutation or key of the 10 positions is:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 7 & 5 & 8 & 1 & 10 & 3 & 2 & 9 & 4 & 6 \end{pmatrix}$$

For messages longer than 10 letters, the same permutation is repeated for each block of 10 letters. The letter frequency approach will not be successful in breaking a transposition system since the letters are not substituted. Suffice it to say, however,

Figure 1. *The Enigma Machine*

that other techniques are available that allow a simple transposition system such as the above to be broken.

One further interesting historical cryptosystem will be mentioned: the *Vigenère cipher* [2]. Using this system first devised in the 16th century but later attributed to Vigenère in the 19th century, the message is written out and a keyword chosen which is written letter by letter under the plaintext message, and then a Caesar shift operation performed on each letter. For example, if the message is "FOURSCORE AND SEVEN YEARS AGO " and the key is the word "GETTYSBURG", the encryption would be:

Plain	F	O	U	R	S	C	O	R	E	A	N	D	S	E	V	E	N	Y	E	A	R	S	A	G	O
Key	G	E	T	T	Y	S	B	U	R	G	G	E	T	T	Y	S	B	U	R	G	G	E	T	T	Y
Cipher	M	T	O	L	R	V	Q	M	W	H	U	I	M	T	U	X	P	T	W	H	Y	X	U	A	N

The purpose for introducing the Vigenère approach is to provide the basis for the provably most secure cryptosystem--albeit one that is essentially the most expensive, the "one-time pad". One may think of the one-time pad as a Vigenère cipher where the key is unending, with each letter being randomly chosen. Using the same example as above, an example of a one-time pad might be:

Plain	F	O	U	R	S	C	O	R	E	A	N	D	S	E	V	E	N	Y	E	A	R	S	A	G	O
Key	U	T	Y	O	M	G	E	F	E	M	X	F	H	K	F	A	D	A	H	R	A	B	E	H	D
Cipher	A	J	T	G	F	J	T	X	J	N	L	J	A	P	B	F	R	Z	M	S	S	U	F	O	S

Electromechanical encryption

Now let's fast-forward a few hundred years. One distinct difference in cryptosystems beginning in the 20th century was the introduction of electromechanical devices that allowed for a semiautomated production of text and cipher.

Undoubtedly the best known such devices was the Enigma machine, used by Germany in the Second World War– and the success of the British at Bletchley Park in breaking the Enigma codes was one of the most important events of that conflict. [3]

Still, the algorithm driving the Enigma machine and all of its relatives was essentially a variant of the Vigenère cipher using extremely long keys. (It is also worthy of note that these cipher machines used so extensively in World War II received their first patent in the United States in the 1920s.)

During the 1940s, in addition to the war effort, technological strides were made that resulted in the first digital computers. It soon became apparent that cryptology, as well as many other scholarly fields, would soon adopt the computer as an essential tool, and in the case of cryptology, that once and for all we would begin to think of our underlying symbol set as the aforementioned binary or {0, 1} alphabet.

Lucifer and its descendants

In 1973, Horst Feistel of IBM published a cryptosystem called *Lucifer* [4] designed for the computing world that combined the principles of substitution and transposition as illustrated above. Lucifer inspired other cryptosystems, and by 1977, researchers at IBM, the National Security Agency, and the National Bureau of Standards (now called the National Institute of Standards and Technology) published the *Data Encryption Standard* for the United States, commonly called *DES* [5].

DES is no longer recommended for usage, but it had a 25 year run before it was generally considered to be broken. However, because many of the principles behind the DES are still used, it is worth a cursory examination of this cryptosystem (Figure 2).

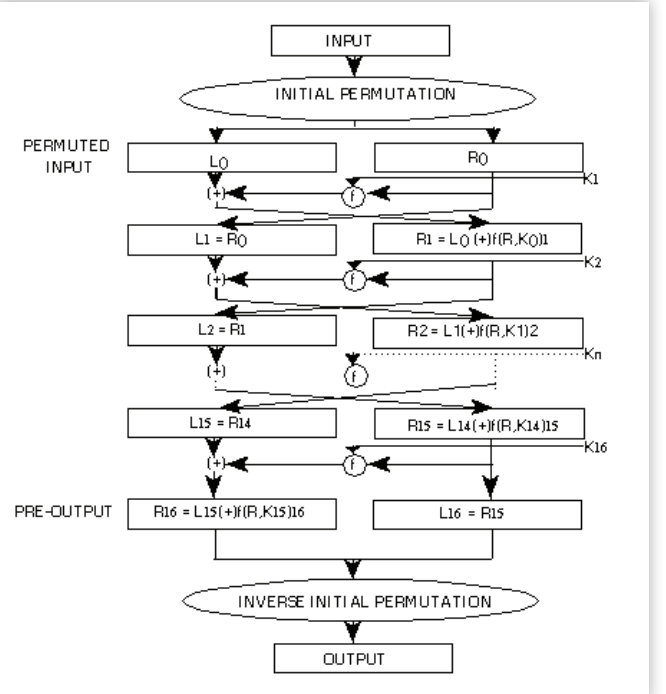


Figure 2. *The Data Encryption Standard Algorithm*

First, DES used as a block cipher was designed to transform each 64 bits of a message into a 64-bit cipher. Furthermore, the key could be any 56 bit string and so the potential number of keys was 2^{56} = 72,057,594,037,927,936. All of the operations constituting the DES were either exclusive-OR (XOR) operations, thus substitutions; or table lookups, hence transpositions.

DES consisted of 16 mappings, or rounds. Without burdening the reader with the full detail of the DES encryption, the idea is that the 64-bit plaintext is split into a left half (L_0) and a right half (R_0), then in each of 16 successive rounds, the next left half (L_j) is simply the previous right half, R_{j-1}; whereas the next right half is the former left half XORed with a more complex transformation. Algebraically, the transformation can be written as

$$L_j = R_{j-1}; R_j = L_{j-1} \oplus f(R_{j-1}, K_j).$$

The K_j is a selection of a certain number of bits of the 56-bit key. The function f is composed of three parts, described as

$$f(R_{j-1}, K_j) = P(S(E(R_{j-1}) \oplus K_j)),$$

where E is a simple (and constant) injection of the 32 bits of R_{j-1} into a 48-bit string, which is then XORed with selected 48 bits of the key. S is a specific mapping (called the S-boxes) of the 48 bits down to 32 bits, and P is a fixed permutation of the resulting 32-bit string. A complete source code implementation of DES can be found in [1].

Virtually everyone who has studied DES has agreed that all of the security in the DES is contained in the S-boxes.

There are several critical aspects of the DES transformation. First, the operations consisting of XORs and memory references are all extremely fast at the machine level. Next, from the diagram above, the fact that the left and right halves switch does not occur in the last round enables the decryption algorithm to be essentially the same as the encryption

algorithm, with only the selection of key bits differing. In the context of 1970s memory availability, the fact that the same algorithm could serve both for encryption and decryption implied an important saving of code, whether in software or hardware. Finally, with the technology of the 1970s, exhaustive search was essentially infeasible; however, by the mid-1990s, techniques had developed (*differential cryptanalysis* [6]) that put DES in jeopardy.

Throughout its 25 year lifespan, DES served an important role. However, it was always a subject of great debate within the cryptography community as to the criteria in determining the various components of the DES transformations, particularly the design of the S-boxes. Nevertheless, at the end of the day, one fact remained: DES was fast and efficient, since the operations were simple at the machine level.

When the United States government decided to replace DES as a standard, the approach taken was designed to be far more transparent than in the development of DES. An international competition was conducted over several years, with the result in the fall of 2001 being the adoption of the current *Advanced Encryption Standard*, or *AES*, which is essentially the encryption system devised by Joan Daemen and Vincent Rijmen and known as *Rijndael* [7]. Although Rijndael uses some interesting properties of Galois Fields, it retains the principle of the usage of substitution and transposition, with the corresponding operations still being very fast. In addition, Rijndael or AES could be constructed with several sets of possibilities for key length, going from 128 bits to 256.

However, AES, like DES and all of the other methods described above, still had one critical limitation. The sender and the receiver of messages under AES required to possess identical key information in order to both encrypt and decrypt. In the days of a single sender and receiver, this might not have been too great a limitation. However, in our current environment, it is not unreasonable to think of a network of a thousand users. In such a case, if each pair of users were to require separate keys for communication thenw $\binom{1000}{2} = \frac{1000 \times 999}{2} = 499{,}500$ keys would be required!

There is yet another limitation of symmetric systems. Perhaps as important as encryption in today's electronic society is the need for authentication--the method by which the creator of a message can prove that he or she alone could have created the message. In the paper era, this requirement is familiar to all in many ways, not the least of these being the checkbook.

No symmetric system can ever be used for authentication, or what is also called digital signature since there is always another party with the identical key information as the person wishing to authenticate.

Asymmetric cryptography

Around the time of the introduction of the DES, Diffie and Hellman [8] described a model by which the key management problem as described above could be solved. Their concept was to suppose that it could be possible for a key K to have two components, a public part that we will call K_p and a secret part that we will call K_s. Thus the entire key could be described as $K = (K_p, K_s)$. We would furthermore require that only the public part of the key, K_p, would be necessary for encryption, but the entire key K would be necessary to decrypt.

Should we be able to find such a key, it is easy to see how a cryptosystem comprised of such keys would solve the key management problem. Namely, each user would create his or her own K, and place the public part in an open directory. Then, the user would look up my public-key and encrypt the message using it should he or she wish to send a message to me. Then, only I will be able to decrypt the message since only I will possess knowledge of the secret key. Consequently, one never has to distribute keys and the key management problem is solved.

However, what Diffie and Hellman described was only an idea. It was left to the crypto community to find a workable model of such a "public key" cryptosystem. We would also call such a system asymmetric since the sender and receiver each possess different information. By contrast, in all of the cryptosystems described to date, the sender and receiver have exactly the same information and so all of the preceding cryptosystems are referred to as symmetric.

Several models were described as candidates for a public key cryptosystem, many being based on the classical combinatorial knapsack problems [1]. Most of these were demonstrated to be breakable by the work of Lagarias and Odlyzko [9] and independently by Brickell [10]. However, within a short period of time, Ronald Rivest, Adi Shamir, and Len Adleman published a remarkable description of a workable public-key cryptosystem based on the difficulty of factoring large numbers, that they called the RSA public key cryptosystem [11].

The essence of the RSA cryptosystem is that it is computationally feasible to factor most numbers of several hundred decimal digits. The complete RSA cryptosystem is simple enough to describe that the company later formed by its creators, RSA Security, printed it on a company T-shirt.

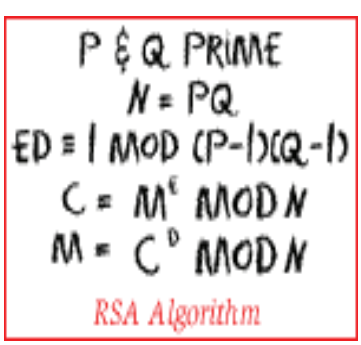


Figure 3. *RSA on a T-Shirt*

To create an RSA, each user of the system chooses two large prime numbers, p and q, each of 200 decimal digits, then computes their product n = p×q, and then their Euler function or totient f(n) = (p-1)×(q-1). Next, a number e is found such that GCD (e, ϕ(n)) = 1, and then d is computed so that e×d ° 1 (mod ϕ(n)). With this done for each user, the cryptosystem is established, and the user's public-key is (e,n) and his or her secret key is (p, q, d). (e, n) is placed in a public directory associated with the user.

The message to be encrypted is broken into blocks so that the number of bits in each block is less than the number of bits when n is interpreted in binary. Suppose each message block is referred to as m and is thought of as an integer. The encryption is simply the computation $c = m^e \pmod n$, and c is the cipher or encrypted message. The sender is able to perform this computation since e and n can be looked up for the eventual receiver in the public directory. The decryption can only be performed by the receiver with his or her secret key, and this computation is $c^d \pmod n$.

Why does this work? The justification goes back hundreds of years to Fermat and Euler [12]. They showed that for any

```
(* The Rivest-Shamir-Adleman Public Key Cryptosystem *)

(* Here is a step-by-step implementation of the Rivest-Shamir-Adelman algorithm, in all its glory, with 400-digit numbers. *)

(* First, select at random a 200-digit number, p, and test to see if it is prime. You may have to do this a number of times. *)
i = 1;
While[! PrimeQ[p = Random[Integer, {10 ^ 200, 10 ^ 201}]], i = i + 1];
Print["i = ", i, "  p = ", p];


(* Now do the same thing to find a prime, q. *)
j = 1;
While[! PrimeQ[q = Random[Integer, {10 ^ 200, 10 ^ 201}]], j = j + 1];
Print["j = ", j, "  q = ", q];

(* Multiply p and q to get n. *)
Print[n = p * q];


(* Compute the Euler function of n, also called the totient,
also called Phi(n). It is the count of how many numbers less than n are relatively prime to n;
when n is the product of two primes, it turns out that Phi(n) = (p-1)(q-1). *)

Print[phiofn = (p - 1) (q - 1)];


(* Now find a number e, which is relatively prime to Phi(n). That is, GCD[e, Phi(N)] = 1. Again, you may have to try over and over again. *)

k = 1;
While[GCD[e = Random[Integer, {p + q, phiofn}], phiofn] > 1, k = k + 1];
Print["k = ", k, "  e = ", e];

(* Now compute the inverse of e, mod Phi(n). This is guaranteed to exist because the GCD[e, Phi(n)] = 1. *)

Print["d = ", d = PowerMod[e, -1, phiofn]];


(* In essence, everything done above constitutes the "key generation" or preprocessing phase of the
   RSA algorithm. Every user of the system would go through the computations above and wind up with a p,
q, n, e, and d. Then, the n and e are the encryption or public keys, and p, q, and d are the secret keys. *)

(* Now, in order for someone to send me an encrypted message, they must be able to look up my public keys,
n and e. Then, if they wish to send me a message, m, they perform the computation m^e (mod n). *)

(* Pick an integer to serve as the message. *)

Print["Message = ", m = Random[Integer, {2, n}]];

(* Now perform the encryption, by computing m^e (mod n) *)

Print["Ciphertext = ", c = PowerMod[m, e, n]];

(* And, going backwards, compute c^d (mod n). *)

Print["The decrypted information, mprime = ", mprime = PowerMod[c, d, n]];


(* Of course, what we desire is that the message m, encrypted to c and decrypted leads back to
   m. We can check that by comparing m and c^d (mod n). Lo and behold, how do they compare indeed? *)
Print["AND THE ANSWER BETTER BE ZERO!!!!!!!"]


Print["THE ANSWER IS ....", m - mprime]
```

Figure 4. *RSA in Mathematica*

$a \neq 0$ in a modular system, $a^{\varnothing(n)} \equiv 1 \pmod{n}$. We should also note that since $e \times d \equiv 1 \pmod{\phi(n)}$, $ed = k\phi(n) + 1$ for some k. Then, to prove the correctness of RSA, simple algebra shows:

$$c^d = \left(m^e\right)^d = m^{ed} = m^{k\phi(n)+1} = m^{k\phi(n)} \times m^1 = (m^k)^{\phi(n)} \times m = 1 \times m = m \pmod{n}.$$

with the last step using the result of Fermat and Euler.

A full implementation in Mathematica is attached. A Sage version by the author is also publicly available at [13].

As simple as is the description of RSA, there is a price to be paid. The sole operation in the encryption and decryption steps are exponentiation of integers and reduction modulo another integer. Breaking this down further, exponentiation is repeated multiplication, and reduction modulo n is integer division, which is essentially equivalent at the machine level to multiplication.

There is also a standard for authentication, known as the Digital Signature Standard or DSS. As with RSA, the algorithm involved in computing the signature also involves exponentiation in a similar fashion to the RSA.

The dichotomy

Virtually all of the symmetric systems we have described can be implemented in a way that is computationally efficient and they are therefore well suited to crypto applications that involve large data sets–for example sound, image or video files.

But in large networks, the key management problem can pose severe challenges–and furthermore, no symmetric system can be adapted to use for authentication.

On the other hand, virtually all asymmetric systems have a computational overhead involving operations like multiplication that are orders of magnitude slower than those involved in symmetric systems. Thus cryptology awaits the discovery of the technique for asymmetric cryptosystems that will have the efficiency and throughput of current symmetric systems.

Faster multiplication

It is not to say that the speed of multiplication cannot be improved. Clearly, the algorithm we learned in elementary school to multiply integers requires n^2 multiplications when the numbers are of n decimal digits. For example, this problem requires $9 = 3^2$ separate multiplications:

```
                4   6   3
            ×   7   2   9
            ---------------
                4   1   6   7
            9   2   6
        ---------------------
    3   2   4   1
```

Yet the faster methods of multiplication, for example [14] carry with them other disadvantages. The author and others have explored this with a technique [15] that can improve speed up on parallel machines. Nevertheless, it remains the case that multiplication when compared to XOR always loses.

Will we ever see the merging of the algorithms for symmetric and asymmetric cryptology and authentication? Who can say? But it is a worthy challenge to pursue.

REFERENCES

- [1] Patterson, Wayne, *Mathematical Cryptology,* Rowman and Littlefield, 318 pp, 1987.
- [2] *http://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher*
- [3] Welchman, Gordon, *The Hut Six Story: Breaking the Enigma Codes,* McGraw-Hill, New York, 1982.
- [4] Feistel, Horst, *Cryptography and Computer Privacy,* Scientific American, Vol. 228(5), May 1973, pp. 15-23.
- [5] National Bureau of Standards, *Data Encryption Standard,* FIPS PUB 46, Washington, January 1977.
- [6] Biham, Eli and Adi Shamir, *Differential Cryptanalysis of the Full 16-Round DES,* CS 708, Proceedings of CRYPTO '92, Volume 740 of Lecture Notes in Computer Science, December 1991.
- [7] Daemen, Joan and Vincent Rijmen, The Design of Rijndael, Springer-Verlag, Berlin, 2002.
- [8] Diffie, Whitfield and Martin Hellman, *New Directions in Cryptography,* IEEE Trans. on Information Theory, Vol. IT-22, 1976, pp. 644-654.
- [9] Lagarias, Jeffrey and Andrew Odlyzko, *Solving Low-Density Subset Sum Problems,* Journal of the Association for Computing Machinery, January 1985.
- [10] Brickell, Ernest, *Are Most Low Density Polynomial Knapsacks Solvable in Polynomial Time?*, Proc. 14th Southeastern Conference on Combinatorics, Graph Theory, and Computing, 1983.
- [11] Rivest, R.; A. Shamir; L. Adleman (1978). *"A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM* **21** (2): 120–126. DOI:10.1145/359340.359342
- [12] *http://en.wikipedia.org/wiki/Fermat%27s_little_theorem.*
- [13] *http://www.sagenb.org/home/pub/2906/.*
- [14] Karatsuba, A. and Yu. Ofman, *Multiplication of Many-Digital Numbers by Automatic Computers,* Proceedings of the USSR Academy of Sciences **145**: 293–294, 1962.
- [15] Abdelguerfi, Mahdi, Dunham, Andrea, and Patterson, Wayne*, MRA: A Computational Technique for Security in High-Performance Systems*, Proceedings of IFIP/Sec `93, International Federation of Information Processing Societies, World Symposium on Computer Security 1993, May 1993, pp. 381-397.

DR. WAYNE PATTERSON

Dr. Patterson is Professor of Computer Science at Howard University. He has also been Director of the Cybersecurity Research Center, Associate Vice Provost for Research, and Senior Fellow for Research and International Affairs in the Graduate School at Howard. He has also been Chair of the Department of Computer Science at the University of New Orleans, and in 1988 Associate Vice Chancellor for Research at that university. In 1993, he was appointed Vice President for Research and Professional and Community Services, and Dean of the Graduate School at the College of Charleston and the University of Charleston, South Carolina. In 1998, he was selected by the Council of Graduate Schools, the national organization of graduate deans and graduate schools, as the Dean in Residence at the national office in Washington, DC. His other service to the graduate community in the United States has included being elected to the Presidency of the Conference of Southern Graduate Schools, and also to the Board of Directors of the Council of Graduate Schools. Dr. Patterson has published more than 50 scholarly articles primarily related to cybersecurity, and a leading textbook, Mathematical Cryptology (Rowman and Littlefield, 1986). He has been the principal investigator on over 35 external grants valued at over $6,000,000. In August 2006, he was loaned by Howard University to the US National Science Foundation to serve as the Foundation's Program Manager for International Science and Engineering in Developing Countries.

TIMING ATTACK AGAINST THE CBC OPERATING MODE

MATTHIEU BONTROND

Early 2000, a really nice job has been performed by the EPFL Security and Cryptography Laboratory to study the security of the TLS/SSL protocol. In 2001 and 2002, Serge Vaudenay started to warn the scientific community on security flaws induced by the CBC padding in various security protocols (ref 1 & 2). In 2003, a practical attack has been implemented in a joint work with some of his students and collaborators (ref 3). The CRYPTO'03 presentation provides a clear explanation of the process and description of the limits.

This attack enables decryption of blocks without attacking the encryption key. Compared to classical cryptanalysis techniques, the amount of computation and trial and error attempts are very reasonable. These properties may render this attack very efficient against SSL-like protocols; however today's protocols' configuration have a great impact on this attack.

This attack exploits properties of different components which are often implemented in communication protocols:

- A characteristic of the CBC operating mode and weak padding types,
- An inherent property of the "MAC then Encrypt" philosophy,
- Error messages produced at a decryption error event.

"MAC then Encrypt" philosophy

The "MAC then Encrypt" philosophy consist in computing some integrity check value on data to be sent before the ciphering process. This is a widely adopted process but the reverse process "Encrypt then MAC" is more relevant in term of security because information leakage will occur potentially less easily.

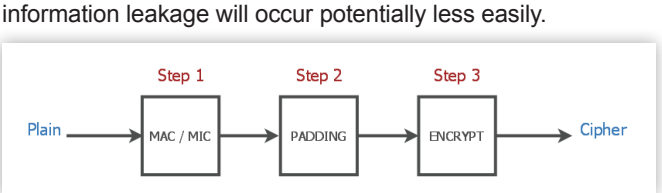


Figure 1. *"MAC then Encrypt" encryption process*

The reverse process takes place in the reverse order, thus the validity of the padding scheme is checked before data integrity.

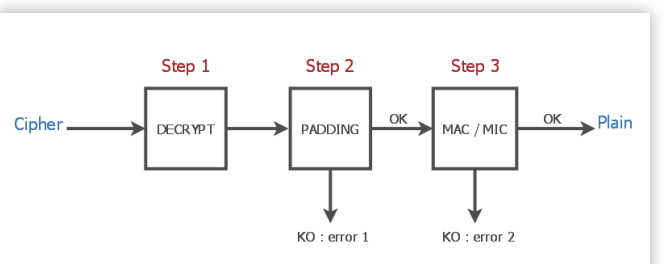


Figure 2. *common "MAC then Encrypt" decryption process*

As a consequence, potential error messages may be produced at different stages. In particular integrity check require more resources to be computed than the padding scheme to be controlled. This holds even more if the data integrity is ensured by a strong cryptographic function. This remark is an entry point to perform a timing attack. The attack exposed by the researchers consists in exploiting the possibility to differentiate padding errors and integrity check failures.

Error messages

When this problem has been discussed between people from OpenSSL team and security researchers of EPFL to implement

countermeasures, some people suggested to encrypt error messages or to provide only one identical error message for both errors. Both countermeasures cannot resist to timing analysis since the MAC computation, which occurs after padding control, will still take time. Researchers showed that statistical analysis of the response time can provide information on the operation being executed. The immediate simple countermeasure suggested was to check the integrity value even if the padding has not been properly set. Nevertheless this countermeasure will not resist to an attacker who has access to logs files. He will have the opportunity to learn which kind of error has been produced during the decryption process. Measuring the size's evolution of the logs files might be potentially sufficient.

Padding Schemes

Padding errors are produced when the format of the padding does not meet the one expected by the underlying cryptographic mechanism. Such error is also often a consequence of a corrupted ciphertext. There exist several padding scheme commonly used. All these schemes allow having a data input size that can be divided by the block size of the block cipher. Usually the last byte of the last decrypted block indicates how many bytes to remove to get the original message. The most common padding schemes are described in the table below:

AINSI X.923	AD 09 7C 00 00 00 00 **05**
	AD 09 7C 01 02 03 04 **05**
PKCS7	AD 09 7C 05 05 05 05 **05**
ISO 10126	AD 09 7C D5 A7 8C 13 **05**

Figure 3. *References to common padding schemes*

The ISO 10126 padding scheme specifications requires to set the last byte to n where n is the number of byte to remove to get the original message, and to fill the remaining (n-1) bytes with random data. This is the only non-weak scheme presented here. Unfortunately this is not the most common scheme because there is random data to compute. This attack will not allow the decryption of a full block with this padding. Nevertheless it will be possible to learn the value of the last byte (hence the length of the plaintext) which is sometimes a great source of information. The attack will work fine with the three other padding types because the content of the padding is known by the attacker.

Operating modes

Block ciphers algorithms require also to be used with an operating mode. Various works have been performed around operating modes providing authentication of the underlying data. Nevertheless they are still not widely deployed and some communication protocols use older operating modes. One of the most common operating modes is the CBC mode (Cipher Block Chaining). In particular, this operating mode is commonly used with the DES/TDES encryption algorithm. Despites a drawback inherent to the chaining operation, this operating mode is simple and no flaws have been reported. S. Vaudenay has shown that even if this operating mode is not flawed, it may induce a flaw in a communication protocol. This example emphasises the fact that the security of a whole system cannot be only measured by a separate and careful analysis of each of his component. Their interactions are part of the analysis to lead. The CBC mode basically consists in ciphering the current plaintext XORed with the previous ciphertext (XOR is the "exclusive OR" operation). An initial vector is used as a previous ciphertext to cipher the first plaintext block. This vector is often a public value.

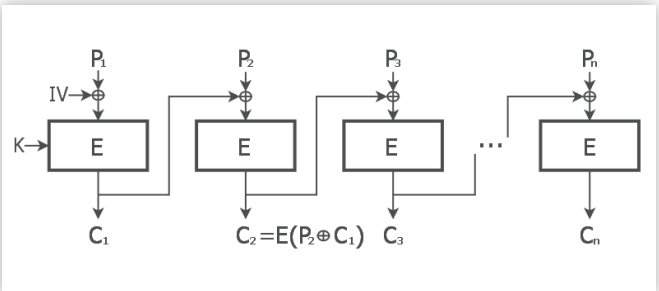


Figure 4. *overview of the CBC operating mode*

C_i values are made public since they are ciphered. Thus an attacker can collect those blocks and try to decrypt some of them. Let's consider that the second block C_2 contains an interesting value, for instance a password. For the purpose of the description of the attack, we will try to decrypt this particular block.

Now, we suppose that we want to decrypt the block C_2. The operating mode used is the CBC and the padding scheme PKCS7. The encryption function does not matter, E is set to DES.

The oracle model

To pursue the attack, one needs to have access to the decryption process of one of the protocol's protagonist. This is not an easy task because decryption error usually causes fatal error. Once the connection is closed it will not be possible to submit any ciphertext for decryption. A new connection will be opened with a new session key. Then it will not be anymore possible to decrypt C_2. This is what happens concerning the TLS protocol and as a consequence this attack won't work against the TLS protocol. To work fine, it is essential to exploit too permissive protocol configurations.

Thus an attacker needs to submit at least two blocks of data. We denote C_1' the first block and C_2 the second block. C_2 is the block to be decrypted and C_1' is set to randomly. The decryption process will produce two plaintexts. The first one, P1', contains random data and the second one P_2', is combination of the original P_2, C_1 and C_1'. C_1 and C_1' are public ciphertext values.

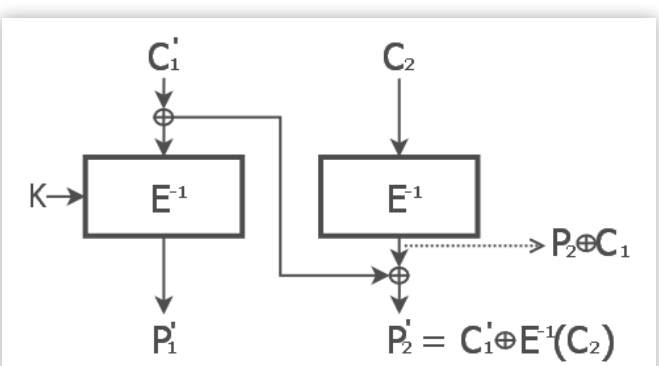


Figure 5. *decryption operation of C_1' and C_2*

Then, the decryption appliance will check whether P_2' is a valid padding. Considering the PKCS padding mode, P2' should look like random value then n bytes set to n:

P_2' = AD 09 7C E3 04 04 04 **04 ?**

If the padding is valid then the MAC is computed. Once a valid padding is detected, then all corresponding bytes may be deduced by a simple XOR operation. To decrypt a full block, C_1' shall be modified following a trial an error process. To produce the padding value 07 07 07 07 07 07 07 07, we will try first to produce the padding XX XX XX XX XX XX XX 00**.** To get such result, only the last byte of C_1' will be modified. Then the two blocks (C_1' and C_2) will be submitted for decryption. This process is repeated until we can ensure that there is no padding error. The first step is to increment the last byte of C_1' until a valid padding is produced. Keep in mind that there is only a XOR operation between C_1' and the decrypted of C_2. Thus modifying C_1' step by step will necessarily lead to a valid padding. At this step, the appliance will have decrypted:

P_2' = AD 09 7C A3 87 55 3D **00**

To check this hypothesis, the previous byte may be incremented with no effect. Otherwise a larger valid padding has been detected and the same process applies to the previous byte and so on to discover the length of the padding. Once a valid padding is detected, all the C_1' "padding bytes" are incremented and the previous byte is modified until a valid padding is obtained:

P_2' = AD 09 7C A3 87 55 3D **01**
ξ ?
P_2' = AD 09 7C A3 87 55 **01 01**
ξ ?
P_2' = AD 09 7C A3 87 **02 02 02**

Once again this work fine because there is only a XOR operation between C_1' and $E^{-1}(C_2)$. For each byte, an attacker will successfully find a correct value with an average probability of 1/128. The full decryption of one block will require in average 1000 queries and measures. The cost of this attack is particularly cheap in term of resources compared to classical cryptanalysis techniques. The decrypting appliance takes the particular role of an oracle, answering yes or no depending if the padding of the submitted input is valid. We can then use the described process to produce the desired padding. This leads to a situation where all values may be deduced by the attacker.

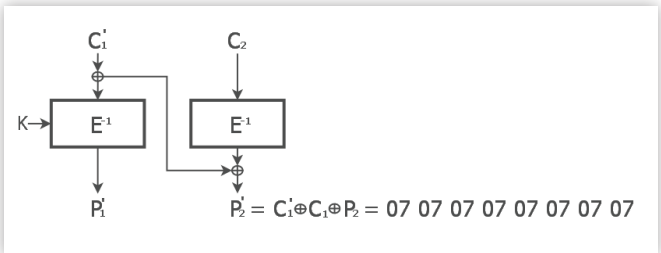


Figure 6. *Expected padding to deduced the value of P2*

Finally P_2 is the result of the XOR operation between C_1, C_1' and 07 07 07 07 07 07 07 07. This process has to be repeated for each block to decrypt. It is better to identify interesting block of data first.

Conclusion

This attack is rather simple and efficient. Even if the protocols configuration is particular, this remains today a very common configuration in communication protocols. The crucial point is the ability to get knowledge on the decryption process during a sufficient amount of time. The protocol has to be permissive to allow various decryption attempts for a given session. Time response analysis is a powerful tool to differentiate errors types. Maybe that other kind of implementation might be imagined.

References:

- [1] 2001: Vaudenay S., *CBC Padding: Security Flaws in SSL, IPSEC, WTLS, ...*
- [2] 2002: Vaudenay S., *Security Flaws Induced by CBC Padding Applications to SSL, IPSEC, WTLS*
- [3] 2003: Canvel B., Hiltgen A., Vaudenay S., Vuagnoux M., *Password Interception in an SSL/TLS Channel*

radware

Protecting Networks from a New Age of Hacktivism

Radware Attack Mitigation System:

- Real-time, Multi-vector Attack Detection
- Hardware Accelerated DDoS Mitigation
- Integrated Event Correlation & Reporting
- 24x7 Emergency Response Team Support

For more information, visit: www.radware.com

AUTOMATED ALGEBRAIC CRYPTANALYSIS

THEODOSIS MOUROUZIS

Abstract – Algebraic attack is a form of known-plaintext attack and consists of two basic steps. Firstly, one converts the given cryptographic primitive into a multivariate system of polynomial equations usually over or any other algebraic system and then tries to solve for the secret key. The first step is called *modelling* while the second step *solving*.

Crypto-designers' aim is that the underlying system of equations is not solvable faster than exhaustive key search. In general, solving a random multivariate system of equations is *NP-hard* [11]. However, in most cryptographic schemes, their rich algebraic and geometric properties can be further exploited to solve the underlying system.

In this article, we provide an introduction to algebraic cryptanalysis and we describe how this 2-step process can be considered as an automated cryptanalytic process. Such attacks have been a big success for stream ciphers, however for block ciphers, until recently, only a limited number of rounds could be broken. In the last section we present a key recovery algebraic attack for 4 rounds of the Russian government standard block cipher GOST [7] given 2 known pairs of plaintexts and ciphertexts [13].

Keywords – Cryptanalysis, algebraic attack, NP-hard, Multivariate System, Algebraic Normal Form (ANF), Conjunctive Normal Form (CNF), Multivariate Quadratic (MQ), SAT, GOST block cipher

Introduction

Cryptanalysis of block-ciphers is divided into two main classes; the **structural attacks** and the inversion or **generic attacks**. Structural attacks exploit the particularities of a cipher due to its design and the specific properties of its underlying components. Generic attacks are form of black-box attacks and are general purpose algorithms that solve multivariate systems of equations. If we manage to solve this very complex system of equations and obtain the secret key , then we launched a successful algebraic attack against the system. Algebraic attacks apply to a variety of ciphers, ranging from blockciphers, like AES and Serpent [2], to stream-ciphers, like Toyocrypt [9] and Bluetooth [8], and asymmetric cryptosystems, like HFE [10].

Algebraic Cryptanalysis is a subfield of cryptanalysis, whose success relies on the fact that some block ciphers exhibit a high degree of algebraic and geometric structure. It is a *known-plaintext* attack and consists of the following two steps:

Step 1: (MODELLING)

Express the cipher operations as a multivariate system of polynomial equations over or any other algebraic system in terms of key plaintext and ciphertext bits.

$$\{f_1(K,P,C)=0, f_2(K,P,C)=0,..,f_r(K,P,C)=0\} \Leftrightarrow E(K,P)=C$$

,where are functions describing cipher's operations.

Then substitute all known pairs in order to decrease the complexity of the system by eliminating some variables, resulting in equations involving only bits of the secret key For example if we are given one known pair and we can form equations in the key bits , then given another pair we obtain equations in the key bits , increasing the probability that the system can be solved. We assume that the encryption is executed under the same key for the given pairs.

Step 2: (SOLVING)

Solve the underlying multivariate system of polynomials and obtain the secret key.

The idea of algebraic cryptanalysis is not new. Shannon in his paper "*Communication theory of secrecy systems*", states that "***Breaking a good cipher should require as much as solving a system of simultaneous equations in a large number of unknowns of a complex type***",[1].

In general, solving a random multivariate system of equations is known to be NP-hard and thus it is not surprising that all ciphers can be expressed into a system of polynomial equations [11]. That does not imply at all that these systems are solvable faster than exhaustive key search. However, not all the multivariate systems are NP-hard and especially in the area of cryptography some algebraic or geometric properties of the ciphers can be further exploited in order to solve the underlying system of equations.

Design of Systems: Cryptosystems which are designed based on the computational hardness of solving a random multivariate system of equations in finite fields are called ***Mul-***

tivariate polynomial cryptosystems. Unlike linear and differential cryptanalysis, the security of cryptosystem against algebraic cryptanalysis does not increase exponentially with increasing number of rounds. Trapdoors for such systems require designing a system of equation which looks random to an adversary but it is not ”fully” random. All such systems are subject to attacks by Grbner basis algorithms [5]. Quantum algorithms do not perform better than the classical problems for solving systems of polynomial equations and for this reason multivariate polynomial cryptosystems are particular promising for post-quantum cryptography.

As a result of trapdoors, many systems are ***sparse*** and ***overdefined*** so the chance of launching a successful algebraic attack to the system increases. Sparsity is a measure which computes the probability that a given coefficient of a given monomial in the system is zero. We expect that the more the sparsity of the system the less the computational hardness of solving the corresponding system. According to Courtois and Pieprzyk [2], the system of equations obtained from ciphers has a very high probability to be sparse, since efficient implementations of real-world systems require low gate counts. By over-defined system we mean that the number of equations is higher than the number of variables. In cases of over-defined systems we expect to be easier to extract the solutions. We can make a system over-defined by substituting known pairs (*P,C*) encrypted under the same key *K*.

Complexity of Systems: The computational hardness of solving a random system of equations depends on two basic factors; the ***sheer size*** of the system of equations and the ***nonlinearity*** in the relations involved.

In the majority of existing ciphers, the system of equations describing the cipher is so large so it is computationally infeasible to solve it directly. Usually, we have a few equations for a very large number of unknowns.

However, non-linearity is the main factor which makes solvability of the system even more complex. Non-Linearity can be achieved mainly in two ways; using S-Boxes and mixing operations from different algebraic systems. S-Boxes are look-up tables which contain non-linear data and are used in many ciphers for enhancing its *confusion* and *diffusion* properties. Additionally, the fact that operations from different algebraic systems are used in some operations of the cipher contributes drastically in the non-linearity of the system we obtain. For example if both arithmetic and logical operations are used within the cipher then it is not feasible to obtain a linear system of equations either in polynomial or Boolean algebra.

In the rest of this article we provide some more technical details behind the two steps of algebraic cryptanalysis. We describe how we can convert the problem of solving a random system of equations into other problems where there are available software which implement heuristic algorithms searching for solutions, like SAT solvers [12]. We highlight that this 2-step procedure can be converted into an automated procedure or a software attack and we can start experimenting on breaking ciphers even from today.

Expressing the Cipher into Equations

As we explained, *algebraic cryptanalysis* consists of two steps. In this section we provide a detailed explanation on extracting the appropriate algebraic *or logical* representation of the cipher so that we can use them to obtain the key using available software.

The most ”tricky” part in algebraic cryptanalysis is to find the appropriate field to embed the operations into. For example in the cipher AES, we have operations defined both in and However, trying to embed everything into (e.g get the encryption scheme BES) the whole representation becomes very messy. So it is very important to consider every operation over some field where computations are more feasible.

AES was reformulated as multivariate polynomial system of equations *MQ (Multivariate Quadratic)* over [2] and [3]. If any of those systems will be solved faster than exhaustive key search in the future then the AES will be assumed to be broken.

In the rest of this section we describe one way of expressing the cipher into a form, called ***CNF***, which can be efficiently handled by existing software solvers known as ***SAT solvers***. A basic trick to follow is to try to obtain as compact as possible forms of the cipher reducing in this way the complexity of solving step. One way to obtain compact forms is to perform clever optimizations in each step of the conversion.

Conversion Methods

Circuit Complexity is the field of mathematics which studies the representations of Boolean functions as polynomials over the reals. In this section we explain the most widely used representations and corresponding conversion methods. We explain how we can convert a cipher from its reference implementation to a language which can be easily handled by software. For this reason we provide the following very basic definitions.

Definition 1: A Boolean function is a mapping f : {FALSE, TRUE}^{k} →for {FALSE, TRUE} some $k \in \mathrm{N}$.
Any Boolean function can be represented using the operators which define the Boolean algebra and are; ← : Complement, Λ: Conjunction or AND-operator and V :Disjunction or OR-operator.

Additionally, we consider the XOR-operator ⊕ as it is frequently used in the cryptographic schemes. The XOR-operator can be written as $\oplus y = (x\Lambda \leftarrow y)\ V\ (\leftarrow x\Lambda y)$.

Definition 2: **(Disjunctive Normal Form, DNF)**
A Boolean function *f* is said to be in disjunctive normal form if it is a disjunction of clauses, where each clause is a conjunction of literals.

$$\mathrm{V}_{I\subseteq M}(\Lambda_{i\in I}\boldsymbol{x_i}), \boldsymbol{M} = \{\boldsymbol{1},\ldots,\boldsymbol{n}\}$$

Definition 3: **(Conjunctive Normal Form, CNF)**
A Boolean function *f* is said to be in conjunctive normal form if it is a conjunctive of clauses, where each clause is a disjunction of literals.

$$\Lambda_{I\subseteq M}(\mathrm{V}_{i\in I}\boldsymbol{x_i}), \boldsymbol{M} = \{\boldsymbol{1},\ldots,\boldsymbol{n}\}$$

Definition 4: **(Algebraic Normal Form, ANF)**
A Boolean function *f* is said to be in algebraic normal form if it is an XOR of clauses, where each clause is a conjunction of literal.

$$\oplus_{I\subseteq M}\ \boldsymbol{a_I}\Lambda(\Lambda_{i\in I}\boldsymbol{x_i}),\ \text{where}\ M = \{1,..,n\}\ \text{and}\ a_I \in \{FALSE, TRUE\}.$$

Definition 5: **(Polynomial Representation, PR)**
A Boolean function *f* is written in its polynomial form if it has the following representation.

$f(x_1,..,x_n) = \sum_{I \subseteq M} a_I \left(\prod_{i \in I} x_i\right), M = \{1,...,n\}$ and $a_I, x_i \in \{1,0\}$.

In the rest of this section we will show how we can obtain the ANF of a Boolean Function from any of the other forms presented. Firstly we need to define the map *t* ;

$$t: FALSE \rightarrow 0 \; and \; t: TRUE \rightarrow 1$$

The following Lemma explains how we can represent each of the three Boolean operators into algebraic form. We say is the standard representation of *f* if:

$r(t(x_1),..,t(x_n)) = t(f(x_1,...,x_1))$ holds for all possible configurations of $(x_1,...,x_n)$.

Remark: Let *f* be a Boolean function and the standard representation. Then it holds that:

$$f(x_1,x_2) = x_1 \wedge x_2 \Rightarrow r(y_1,y_2) = y_1y_2$$

$$f(x_1,x_2) = x_1 \vee x_2 \Rightarrow r(y_1,y_2) = y_1 + y_2 - y_1y_2$$

$$f(x_1,x_2) = x_1 \oplus x_2 \Rightarrow r(y_1,y_2) = y_1 + y_2 - 2y_1y_2$$

$$f(x) = \leftarrow x \Rightarrow r(y) = 1 - y$$

From ANF to MQ

For a finite field *F* of order *q*, we consider a random system of simultaneous equations in n variables, $G_1(x_1,...,x_n) = y_1$, $l = 1, . . . ,m$,where G_l are polynomials of degree 2 not necessarily homogeneous. Then the problem of solving this system of equations over the given field is called the MQ problem.

Clearly the complexity of finding the solution depends strongly on the parameters *n,m.* As shown in [5], Gro ̈bner bases algorithm are very successful when $m = n < 15$. Courtois and Pieprzyk pointed out that any system of any degree can be written as a 2-degree system [2]. This is proved by using the fact that: $\{m = wxyz\} \Rightarrow \{a = wx;\ b = yz;m = ab\}$

Thus in order to solve a multivariate polynomial system which describes this cipher it is enough to solve the underlying ***Multivariate Quadratic (MQ)*** problem. A classical way to solve a MQ problem is to convert it into a SAT problem and then try to solve it using SAT solvers. In the next paragraphs we explain how we can convert a MQ problem to a SAT problem.

2.2 From MQ to SAT

The procedure to convert a MQ problem to a SAT problem consists of two steps. First, we convert the polynomial system to a Linear System and then we convert the resulting Linear System to a CNF expression. CNF expressions are instances describing SAT problems. More details are given below.

STEP 1: From a polynomial System to a Linear System

Every polynomial is a sum of linear and higher degree terms, assuming that 1 is a variable. We have that the expression (wV ← a)(xV ← a)(yV ← a)(zV ← a)(aV ← wV ←xV ← yV ← z) is tautologically equivalent to a ⇔ (w Λ x Λ y Λ z), or the equation a = wxyz over GF(2). Thus for a monomial of degree d we require d+1 clauses and the total length of those clauses is 3d+1. Thus we can write all polynomial equations into this form.

STEP 2: From a Linear System to a CNF Expression

After performing previous STEP 1, we have now each polynomial is a sum of variables or equivalently a logical XOR. However, long XORs are known to be hard problems for SAT solvers. The sum $(a+b+c+d) = 0$ is equivalent to

$(a \vee b \vee c \wedge d)(a \vee b \vee \leftarrow c \wedge \leftarrow d)(a \vee \leftarrow b \vee c \wedge \leftarrow d)(a \vee \leftarrow b \vee \leftarrow c \wedge d)$; *and* $(\leftarrow a \vee b \vee c \wedge \leftarrow d)(\leftarrow a \vee b \vee c \wedge \leftarrow d)(\leftarrow a \vee \leftarrow b \vee c \wedge d)(\leftarrow a \vee \leftarrow b \vee \leftarrow c \wedge \leftarrow d)$

We summarize the conversion method described in this chapter in the following steps:

1) *Convert the Cipher into ANF*
2) *Convert ANF to MQ*
3) *Convert MQ to CNF*
4) *Convert it to a SAT problem, splitting long XORs*

Solving Multivariate Systems of Equations

In any algebraic system, solving a *linear system* of equations is assumed to be straight-forward provided there are more equations than the number of variables. However, solving a non-linear system of equations is much harder and so cipher designers strive to increase the non-linearity of the underlying operations. Gaussian elimination is used for solving a linear system while for solving a non-linear system, Grbner basis algorithms try to emulate Gaussian elimination for polynomial systems. Unfortunately the computational complexity of Grbner basis algorithms for non-linear systems is no longer polynomial. Additionally, SAT solver is a very powerful tool which can be successfully applied to retrieve the secret key when a compact CNF form of the cipher is provided.

SAT solvers: SAT solvers are sometimes successful in recovering the key bits when we are able to transform the cipher from a compact algebraic form to a CNF form. It is very important that we are able to find optimized representations of the ciphers so that SAT solvers will be more efficient. SAT solvers are heuristic algorithms which try to find (if there exists) a set of assignments of true and false to each of these variables involved in the CNF form so that the entire sentence evaluates as true. There exist many SAT solvers software available such as the MiniSAT v1.13 [4].

In the last section as an example of our methodology we present a key recovery automated algebraic attack for 4 rounds and 2 known plaintext-ciphertext pairs for the GOST block cipher. The software we use can be found from: *http://www.cryptosystem.net/aes/tools/html.*

Algebraic Attacks on Ciphers

In this section we provide some examples on algebraic attacks on given ciphers. Firstly, we demonstrate how trivial is to solve a linear system and then we try to apply our methodology to more complex nonlinear systems.

(A) Linear System

Attacking a linear block cipher is quite straightforward if a sufficient amount of known (*P,C*) is provided. By sufficient we mean that there are as many known (*P,C*) available as needed so that the number of equations exceeds the number of unknowns involved. For example suppose we have a 64-bit block cipher encrypting under a 128-bit key. If there are no known (*P,C*) pairs then we can express the cipher into a system of 64 equations in 128(key bits) + 64(input bits) unknowns, resulting in a system of 64 linear equations in 256 unknowns. However, if one pair (*P,C*) is given, we can substitute it into the previous system of equations resulting in a linear system of 64 equations in 128 unknowns, as only the key bits are not known, which is still not solvable. Plugging, another known (*P,C*) we get 128 equations in 128 variables which is solvable in case the system is linear. However, by plugging more and more known (*P,C*) into the system we can generate more equations, increasing in this way the possibility of solving the system.

(B) Non-Linear Systems

Below we outline how algebraic cryptanalysis can be converted into a software attack combines different software in each stage of the attack.

STEP 1: Write Quadratic equations

It is very important that before we try to extract the quadratic equations of the cipher, we manage to obtain a gate-efficient implementation of the cipher in a sense where the multiplicative complexity of the cipher is minimized as possible. Optimization of the circuit can be achieved using standard techniques found in circuit complexity theory. See [13] for more details regarding optimizations of circuit representations where also an available software is cited.

STEP 2: Convert it to CNF

The Courtois-Bard-Jefferson converter software is available and can perform conversion from MQ to CNF [6].

STEP 3: Solve it

This stage can be done using available SAT solver software. The best SAT solver which be found is the MiniSAT v1.13 [4].

(C) An attack on reduced rounds of GOST block cipher

In this section we provide an example of automated algebraic attack on a reduced round of the GOST block cipher. We perform optimizations in both stages of our attack; conversion and solving stage. More details regarding our methodology can be found in [13].

It appears that we are able to provably minimize the number of non-linear gates in a whole given cipher such as GOST, to a proven lower bound. GOST 28147-89 is a well-known block cipher and the official encryption standard of the Russian Federation. A 256-bit block cipher considered as an alternative for AES-256 and triple DES, having an amazingly low implementation cost and is becoming increasingly popular. We consider the main standard and most widely known version of the GOST block cipher, known as "GostR3411 94 TestParamSet" whose reference implementation as an extension of TLS v1.0. is available as a part of OpenSSL library. The file gost89.c contains eight different sets of S-boxes and is found in OpenSSL 0.9.8 and later: *http://www.openssl.org/source/.*

Now we can encode the whole GOST cipher as follows, which is based on the concept of the multiplicative complexity directly. For more details regarding minimization of the multiplicative complexity see our paper [15], where we develop a methodology for optimizing arbitrary circuits.

1. We will write all the equations algebraically, in the form of their ANF.
2. For the multiplicative complexity we use the exact optimisation obtained above with our SAT solver software.
3. For each input of each multiplication (AND gate) we add one new variable.
4. All the other gates being XORs and NORs, their ANFs gives linear equations over . We did not optimize these linear parts.
5. We get a system of quadratic multivariate equations over *GF*(2) which describes the whole cipher. A ready program to write these equations as a computer file can be found at *http://www.cryptosystem.net/aes/tools.html*. The exact command line is given below.
   For *mod*2^{32} addition, surprisingly, we use our first encoding with 2(*n*- 1) multiplications.

Now we will convert the system of multivariate quadratic equations over that we can obtain using the reference implementation of GOST cipher to a SAT problem and solve it. The results that we present are obtained by following the steps below:

1. We use the Courtois-Bard-Jefferson converter. A Java source code and a working Windows distribution of this program can be found at [6].
2. We use the well-known open-source SAT solver MiniSat 2.06. [4].

Fact 1 (Key Recovery for 4 Rounds and 2 KP): Given 2 P/C pairs for 4 rounds of GOST the 128-bit key can be recovered in few seconds. The memory requirements are very small.

Justification: A ready windows executable program and all the necessary files (one for each S-box), to do just that, together with the solver software, can be obtained from: *http://www.cryptosystem.net/aes/tools.html.*

It also contains a set of 8 optimized files for the S-boxes.

The exact command line option used is: ***axel.exe 1711 4 /ins2 /sat /fix0 /mcom /bard 0 0 1-100***

This will run 100 randomized instances of the problem, call the Courtois- Bard-Jefferson converter [6] and call MiniSat 2.06. [4]. Then it will display various statistics including the median running time for 100 runs which is less than 10 seconds on a modern CPU.

REFERENCES

- [1] C. Shannon, *Communication theory of secrecy systems,* In Bell System Technical Journal 28, pp. 656715, 1949
- [2] N. Courtois and J. Pieprzyk, *Cryptanalysis of block ciphers with overdefined systems of Equations,* Cryptology ePrint Archive, Report 2002/044, 2002
- [3] S. Murphy and M. Robshaw, *Essential algebraic structure within the AES,* In Proceedings Of Crypto 2002, LNCS 2442, pages 116. Springer, 2002
- [4] N. Sorensson and N. Een, *Minisat v1. 13-a sat solver with conflict-clause minimization,* SAT journal pp. 53, 2005
- [5] J. Faugre, *A new efficient algorithm for computing Grbner bases (F4),* Journal of Pure and Applied Algebra 139 pp. 61-88, 1999
- [6] G. Bard, N. Courtois and C. Jefferson, *Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over via SAT--Solvers,* Available at: eprint.iacr.org/2007/02, 2007
- [7] V.Dolmatov, *GOST 28147-89: Encryption, Decryption and Message Authentication Code (MAC) Algorithms,* Cryptocom, 2010
- [8] F. Armknecht, *A Linearization Attack on the Bluetooth Key Stream Generator,* IACR, eprint server, http://www.iacr.org , 2002
- [9] N. Courtois, *Higher Order Correlation Attacks, XL algorithm and Cryptanalysis of Toyocrypt,* Proceedings of the International Conference on Information Security and Cryptography, Lecture Notes in Computer Science Volume 2587 pp. 182-199, 2003
- [10] J. Faugre and A. Joux, *Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Grobner Bases,* Advances in Cryptology - Crypto 2003 2729 pp. 44-60, 2003
- [11] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* W.H. Freeman. ISBN 0-7167-1045-5, 1979
- [12] M. Soor, K. Nohn and C. Castelluccia, *Extending SAT Solvers to Cryptographic Problems,* Lecture Notes in Computer Science Volume 5584/2009 pp. 244-257, 2009
- [13] N. Courtois, D.Hulme and T.Mourouzis, *Solving Circuit Optimisation Problems in Cryptography and Cryptanalysis.* Available at eprint: *http://eprint.iacr.org/2011/475*

THEODOSIS MOUROUZIS

is a PhD student at University College London studying cryptography under the supervision of Dr Nicolas Courtois. His main academic interests lie in the area of optimization of circuit representations and algebraic computations over arbitrary rings for cryptographic and cryptanalytic use and algebraic attacks for hash functions and block-ciphers. He obtained a BSc in Mathematics at University of Cambridge (2005-2008) and a MMaths or Part III (2008-2009) studying Number Theory and Algebra.

CACHE-TIMING ATTACKS ON SYMMETRIC CRYPTOGRAPHIC PRIMITIVES

MICHAEL WISHER

Many attacks on cryptographic algorithms target flaws in the algorithm designs. The flaws can be exploited by intercepting ciphertext, and measuring whether it has statistical biases. From the biases, the attacker assigns probabilities to the different potential keys that generate the ciphertext. Although common in academic literature, this type of attack rarely is practical, since it requires very large amounts of ciphertext generated under a single key. Side-channel attacks consider both the design and implementation of an algorithm. Side-channel leakage is additional information that allows the complexity of an attack to be reduced significantly. Cache timing attacks, t a-c

Side-channel attacks versus theoretical attacks

According to the traditional principles of cryptography, many experts would consider the Advanced Encryption Standard (AES) block cipher, or its Chinese equivalent, SMS-4 to be broken if someone found a statistical attack that could recover a 128-bit key using less than 2^{128} encryptions. Although theoretically broken, the running time of an attack with complexity 2^{120} encryptions is infeasible and the attack could never be completed.

But with access to the cryptographic device, and the ability to precisely inject a single fault into the block cipher chip using a laser, more than 100 bits of the SMS-4 block cipher master key can be easily recovered using just a handful of ciphertexts [1]. The remaining bits of the key can be efficiently guessed.

This style of attack, differential fault analysis, is an active side-channel attack, in which the attacker manipulates the state of the cryptographic device in order to derive the side channel information. The assumptions for this style of attack are very strong – it is possible that someone who stands in front of the device with the ability to manipulate it so precisely is able to read the key directly from its memory. Unless the fault is only transient, the attack is probably detectable.

It is much more difficult to detect passive side-channel attacks, in which the attacker does not affect the device, but uses additional information – such as noise, timing, electromagnetic signals – in addition to the stream of intercepted ciphertext.

Because of the power of this class of attacks, side-channel attacks have recently become a hot topic. Many types of side-channel attack require considerable technical knowledge of the implementation platform, and the details of the attack will vary according to the platform. The topic of this article, timing attacks, mostly allows potential vulnerabilities to be detected during the design process, and algorithm designers can do much to alleviate the vulnerability of their algorithms at the design stage.

Timing attacks

Timing attacks are passive attacks. The side-channel for timing attacks is the difference in the amount of time that it takes to execute different operations or blocks of operations. On some machines, the speed with which a primitive operation can be completed might differ according to the value of the operand.

There might be a difference in the execution time of "y = x <<< 1" and "y = x <<< 5". This difference can lead the attacker to deduce the relationship between x and y.

By the same token, it should be fairly easy to deduce the bit value of k in the following example.

```
if (k == 0) {
    x = x + y;
} else {
    y = y / 3;
    x = x + y *c;
}
```

by measuring how long the operation takes, and comparing it to a profile on the timings of block one (with only an addition), and block two (with an addition, multiplication, and division, the latter of which are historically quite slow).

It is easy to write slow constant-time software, by padding the faster branches with null operations and by avoiding operations that have data-dependant latencies. But it is difficult to write fast constant-time software. This matters for symmetric cryptographic primitives, since they must be optimized for very high throughput.

Cache timing attacks

Cache-timing attacks are a type of timing attack that utilize the difference in timings of indirect addressing.

CPUs are very fast compared to main memory. DDR3 has a latency of around 10 nanoseconds, which means a gigahertz processor that executes a serial algorithm potentially faces bottle-necks as it waits for the data to arrive from the memory.

One solution to this is to position a fast cache between the CPU and the main memory. When the CPU executes an instruction that wants data from memory, the CPU sends a request to the cache. If the data is not present in the cache (a cache miss), it retrieves it from main memory, stores it for future use, and sends it to the CPU. The latency of retrieving the data from the main memory is still present. But if the CPU subsequently requests the same data, and it is still present in the cache (a cache hit), then the latency is reduced to that of accessing the data in the cache.

This opens a side-channel. The attacker can measure the timing of a read from memory. If the timing is small, he can presume the data comes from the cache. Otherwise, he makes the assumption that it comes from main memory. One complication is that, because the cache is small relative to main memory, data is frequently flushed out if it is not used regularly. So it is not guaranteed that a cache hit on one datum will be succeeded by another cache hit on the same datum.

Cache timing attacks and block ciphers

Cache timing attacks apply to symmetric cryptographic primitives – block and stream ciphers - when they use operations that access memory based on secret key material.

They apply to a majority of block ciphers, which since the Data Encryption Standard, have traditionally relied heavily on substitution (s-) boxes. These are operations that implement highly non-linear equations to obscure the relationship between the key and the ciphertext.

Commonly, ciphers use 4x4, 8x8 or 8x32 s-boxes, where an mxn s-box takes an m-bit input and outputs an n-bit output. An mxn s-box can be implemented as n Boolean functions in m terms, but it is more common to implement it as a pre-computed lookup table containing 2^m n-bit entries, since it provides a large gain in efficiency. Whenever an s-box is implemented using a large table, the cipher becomes potentially vulnerable to cache-timing attacks, since each invocation of a lookup table requires access to memory via cache.

While there are few block ciphers that eschew s-boxes, they are in the minority. Daniel Bernstein, the author of one of the most prominent side-channel attacks on the Advanced Encryption Standard, says that using secret data as an array index is a recipe for disaster.

The AES attack

AES is heavily reliant on s-boxes. This block cipher accepts a 128-bit block of plaintext, which it mixes with a key by iterating a round function at least ten times. The round function can be implemented as sixteen table lookups combined with sixteen exclusive-ors. Each table lookup accesses a single 8x32 table for that round.

Prior to the first round, the plaintext is mixed with the round key, byte by byte. In the first round, each of the sixteen table lookups depends only on the combination of a single plaintext byte and a single key byte.

Based upon this, Bernstein implemented and described [2] an attack on a simple server that receives a packet from a client, encrypts the contents with OpenSSL, then sends a receipt to the client. The receipt contains a timestamp that indicates when the encryption has finished. The client gets to choose what is encrypted, but not the key with which it is encrypted.

In the pre-computation phase, the attacker profiles the server with a known key on a target platform, making a list of timings for table lookups using each byte value as an index.

In the online phase, the attacker uses the client to engage the server with an unknown key. By timing the server's responses to handling many plaintexts, the attacker can correlate the timings to select the correct byte from his profile. Then the attacker is able to untangle the key bytes given the known plaintext. Bernstein found he was able to correctly identify the server's unknown key using this method, by sending 227 800-byte packets from the client to the server.

The attack is simple and empirical, but not methodical. In the next section, we show how academics have formalized one type of cache-timing attack.

Prime-then-probe attacks

The cache is shared between many processes, although for reasons of privacy, no process is able to look at cache data of another process. But there is a simple way for to indirectly to observe the moment of data of another process, assuming that there are only two active processes.

Leander, Zenner and Hawkes [3] describe the prime-then-probe attack. The attacking process completely fills the cache with its own data. This causes all of the data of other processes to be flushed out of the cache. The attacker then waits for the cryptographic process to resume.

The cryptographic process, as it executes s-boxes, requests data into the cache on the basis of the indices used in its lookup tables, for example the s-boxes. This table data displaces the attacker's data in the cache.

After the s-boxes have been executed, the attacking process re-reads all of its own data. The attacker measures the time it takes to retrieve this data. Where the latency is small, the attacker's data has remained in those addresses of the

cache. Where the latency matches that of retrieving data through memory, a cache miss has occurred, and it is likely that the cryptographic data used the corresponding addresses in the cache for its table lookups. We assume that the attacker knows the memory address of the lookup table, and is able to map cache addresses to those memory addresses.

Cache lines

For reasons of efficiency, data is not imported into the cache on a byte-by-byte basis, but rather line-by-line.

On modern processors, lines are frequently collections of 64-bytes of adjacent data, and we assume this here-after. If the CPU requests a byte of data, an entire cache-line of 64-bytes will be imported into the cache at once. This reduces the precision of prime-then-probe attacks, as it means that the measurements can determine table indices only to the resolution of the cache line.

Measuring the latency of an s-box lookup into an 8x32 –bit table can provide the high nibble of the index. More generally, the number of bits b leaked by each cache-timing measurement is determined by $b = c - \log2(\gamma_o/d)$, where the table size is 2c entries of d bytes per entry, and γ_o is the size of each cache line in bytes.

Noise

The formalization of the cache-timing attack through the prime-than-probe methodology assumes that the attacker has even more precision than Bernstein's attacker. The benefit of this assumption is that we can reason about the vulnerability of a block or stream cipher abstractly rather than needing to repeat experiments for each platform on which the algorithm can run.

One of the basic assumptions is that the attacker can launch a prime-then-probe attack immediately after each operation performed by the cryptographic algorithm. In other words, the attacker knows when the cryptographic algorithm is going to be given cycles, and can first fill the cache. Then straight after one round of a block cipher, or the emission of a stream cipher's keystream word, the attacker is given priority to re-read his data via the cache.

Another assumption is that there are only two active processes – that of the attacker and the cryptographic process.

Inaccuracies in these assumptions can be modeled as noise. The cure to eliminating noise from the measurements is to repeat them many, many times. Some measurements will give random incorrect key bits. Correct measurements will only give the correct key bits. After sufficiently many measurements, the frequency of the correct key bits will be much higher than for each of the random, noise-induced, erroneous key bits. This is why Bernstein's attack uses so many samples in order to retrieve a single key.

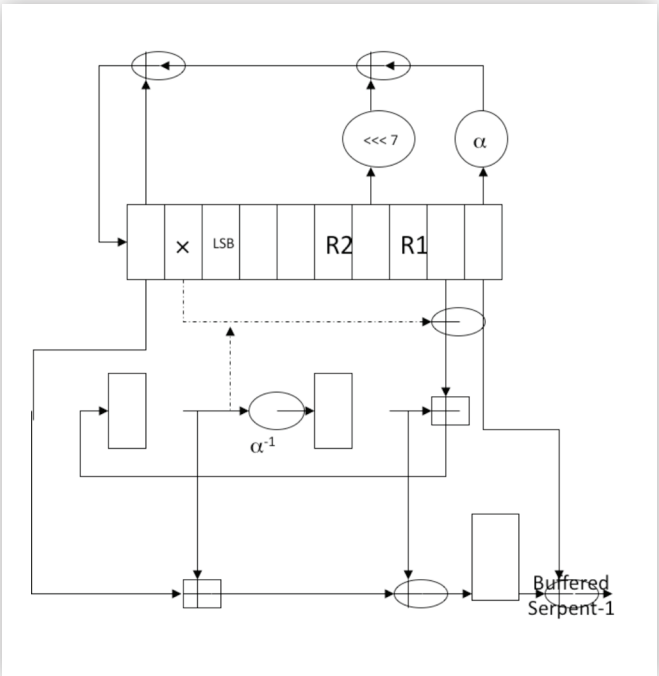
Cache timing attacks and stream ciphers

Even if they don't use s-boxes, stream ciphers can be vulnerable to cache timing attacks. Their defining property is that they maintain state. One method of implementing a state is as a Linear Feedback Shift Register (LFSR).

In software, stream ciphers popularly use word-based LFSRs, where each stage of the register comprises 2n bits for some integer n (eg. a stage size of 32 bits). To diffuse the bits, one or more stages of the shift register will be multiplied by a linear multiplier during each clock. Otherwise, for an autonomous LFSR, at any time bit n of any stage will only depend on a combination of other stages' bit ns.

The Sosemanuk attack

Sosemanuk is an important stream cipher that was declared as one of the four ciphers in the software finalist portfolio of the EU ECRYPT stream competition. Its state consists of an autonomous ten-stage LFSR, and two thirty-two bit registers, R1 and R2, as depicted in the following figure.



Each time the cipher is clocked, to produce a key stream word, one of the LFSR stages is modified by an alpha multiplication (γ_o) and another by its inverse (γ_o-1). These alpha multiplications are implemented using two 8x32 lookup tables.

According the assumptions of the prime-then-probe attack, the attacker is easily able to monitor the upper nibble of each of the values used in the γ_o and γ_o-1 tables, since each is invoked only once during each clock of the cipher. So during one clock, 8 bits of the 320-bit LFSR state are leaked. Over 40 cycles, a system of linear equations can be developed that allows the entire state of the LFSR to be trivially determined.

Once the state of the LFSR has been recovered, the values of R1 and R2 can be determined without the need for using side-channel techniques. The attacker guesses the value of the 32-bit register R1, then uses knowledge of the keystream and the LFSFR to determine R2. He makes up to 232 register guesses, with several bytes of keystream generated for each guess in order to perform verification of the correct state.

Assuming the attacker is able to successfully mount this attack, it is quite shocking that a cipher that offers up to 256 bits of security can be determined so easily. However, there are easy-to-implement countermeasures for this attack.

Without the implementation of counter-measures, similar attacks will apply to all ciphers that use word-based LFSRs.

Defending against cache timing attacks

The best defence against cache-timing attacks is to use a machine without a cache. It will be slower, but there will obviously be no cache leakage. The second best defence is to design cryptographic primitives that use constant-time operations. Bernstein's stream cipher, Salsa20, which uses only addition, rotation and exclusive-or operations, is a good example of this. But only the addition operation is non-linear, and then not offer-

ing high-quality non-linearity. It is possible that advances in differential cryptanalysis could weaken this cipher, leaving it less robust against that attack than others that use a combination of addition and s-boxes.

It is also simple to implement countermeasures for LFSRs, such as the one used by Sosemanuk. Because the tables used by the LFSR are linear, they can be decomposed into smaller tables, and recalculated on the fly. There will be a small (~10%) decrease in efficiency. If the smaller tables can fit into a single cache line each, then the lookups into those tables do not leak information about the index values.

Because the s-boxes are non-linear, they cannot be decomposed. It is rarely practical to implement s-boxes in ways that do not use lookup tables, such as in their Boolean expressions, unless they can be bit-sliced. Smaller s-boxes – eg. 4x4 sboxes - can be packed into a single cache line so don't leak information. But they are weak with respect to other attacks.

One way to mitigate the vulnerability of s-boxes is to invoke the same s-box frequently. If an 8x32 table, occupying 16 cache lines, is invoked sixteen times in a cipher clock, then it will on average displace 10.6 cache lines of the attacker's data. The attacker needs to establish a correspondence between the displaced cache line and the s-box that flushed it. In order to derive the full 64 bits of leaked information, he must check the 16! = 244.3 ways of ordering the information. This reduces the effective leakage to just under 20 bits.

Are cache-timing attacks on stream ciphers realistic?

To average noise out of measurements, the attacker must repeat them many times. In order to do this, he must rekey the stream cipher each time with the same key-IV pair. Otherwise the generated keystream, and sequence of displaced cache lines will be different.

The fundamental rule of stream ciphers is that it can only be used with the same key and IV once. Otherwise it leaks information, since ciphertext C = plaintext P ⅋ κεψστρεαμ K. Ιφ K ρεμαινσ τηε σαμε οωερ συβσεθυεντ ιτερατιονσ, τηεν X1 ⅋ X2 = P1 ⅋ P2. When the plaintexts contain some redundancy, it might be easy to untangle P1 and P2 without knowledge of the key.

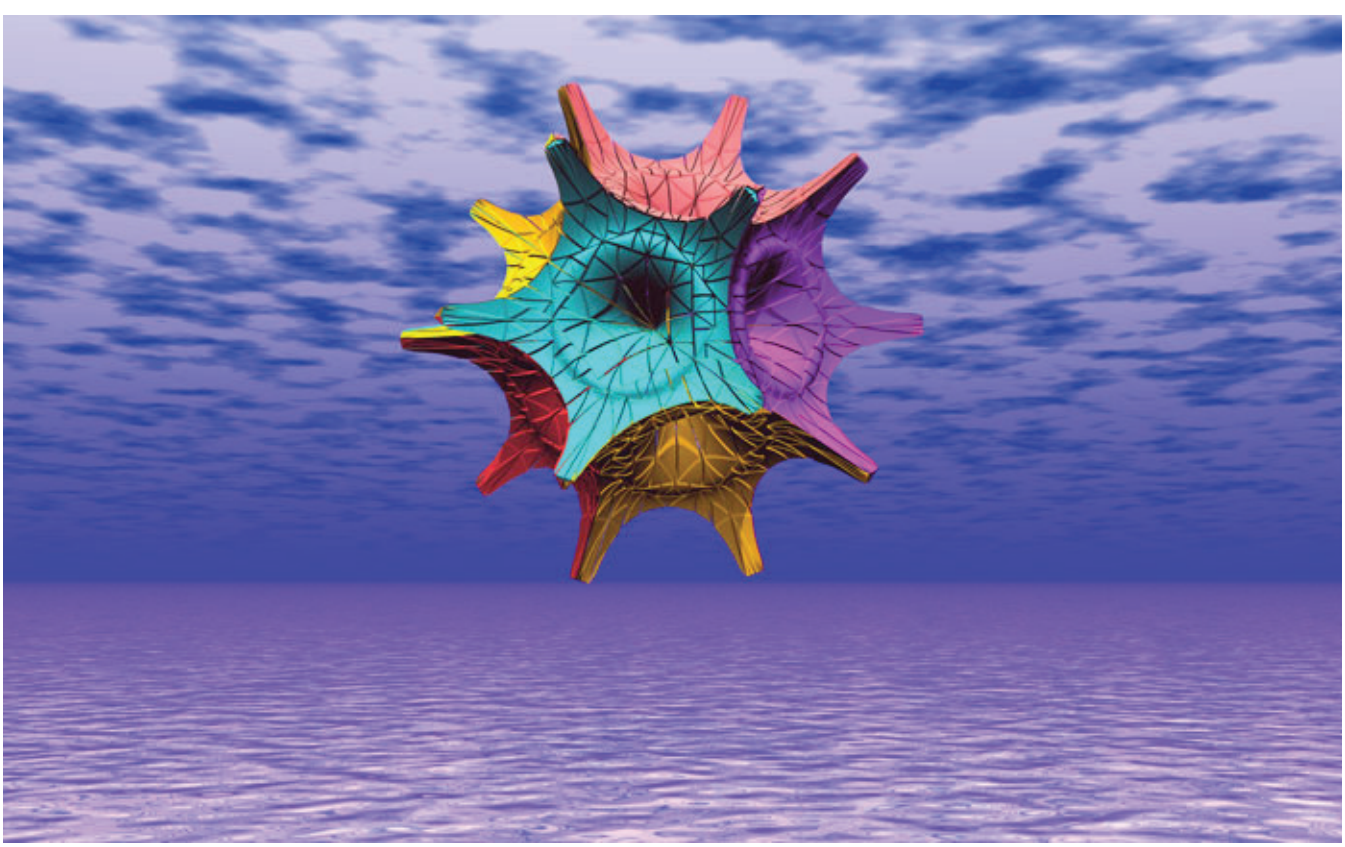
All of the cache-timing attacks on stream ciphers that I have seen acknowledge that multiple measurements under a single key-IV pair must be made. So it seems that this class of attacks is less realistic for stream ciphers than it is for block ciphers.

Conclusion

In this article, we examined cache timing attacks against block ciphers and stream ciphers. As a result of cache-timing attacks, many designers turned away from using s-boxes in block ciphers and stream ciphers. This reduced the 'biodiversity' in modern cipher design. This loss of diversity might lead to problems when advances are made in particular types of statistical cryptanalysis. In particular, I don't view there is very compelling evidence that cache-timing attacks are very effective in practice against stream ciphers, although this is an area that requires more examination.

References

- [1] Ruilin Li, Bing Sun, Chao Li Jianxiong You. Differential Fault Analysis on SMS-4 using a single fault. At *eprint.iacr.org/2010/063.pdf. 2010.*
- [2] Daniel J. Bernstein. Cache-timing attacks on AES. At *http://cr.yp.to/antiforgery/cachetiming-20050414.pdf*. *2005.*
- [3] Gregor Leander, Erik Zenner, Philip Hawkes
- Cache Timing Analysis of LFSR-based Stream Ciphers. Proceedings of Crypto & C-V2



TIMING ATTACKS ON PRACTICAL QUANTUM CRYPTOGRAPHIC SYSTEMS

NITIN JAIN

With photons being the only available candidates for long-distance quantum communication, most quantum cryptographic devices are physically realized as optical systems that operate a security protocol based on the laws of quantum mechanics. But to finally yield a stream of bits (secret key) usable for encryption, a quantum-to-classical transition is required. Synchronization of electronic & optoelectronic components involved in such tasks thus becomes a necessary and important step. However, it also opens up the possibility of timing-based loopholes and attacks.

Introduction

In a letter to Max Born written in 1926 [Born, 1969], Albert Einstein remarked: "Quantum mechanics is certainly imposing. But an inner voice tells me that this is not yet the real thing. The theory says a lot, but does not bring us any closer to the secrets of the Old One. I, at any rate, am convinced that He is not playing dice." This quote, particularly the last part about God not playing dice, indicates Einstein's unwillingness to accept a fundamental tenet of quantum theory: with regards to values of physical quantities, only statistical assertions can be permitted. Indeed, Einstein and some other prominent scientists were also inclined towards the more classical view of the world in which physical systems could be ascribed properties that existed irrespective of whether they were being measured or not [EPR, 1935]. It was thus believed by some that quantum mechanics could not provide a complete description of Nature.

Fast forward to the next century, and with principles of quantum mechanics having been verified in innumerable different experiments, it seems that the earlier view of those scientists was incorrect. Nonetheless, due to its bizarre nature and ideas, quantum mechanics still confounds anyone who tries to understand it. But thankfully, that hasn't stopped us from

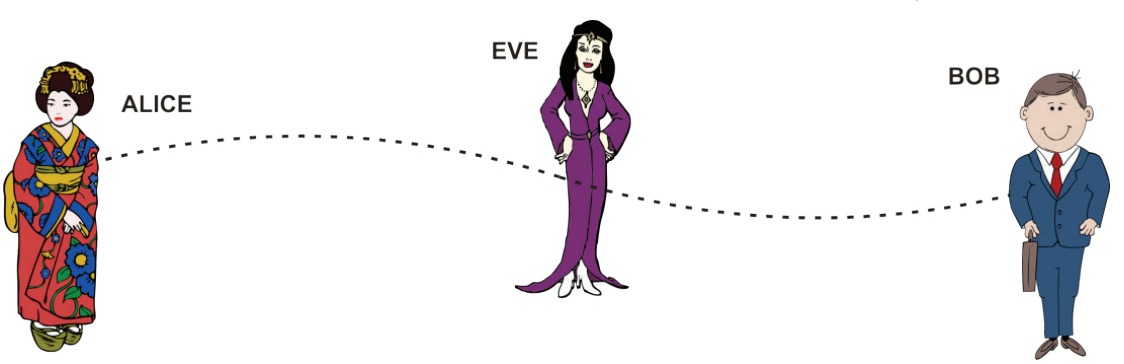


Figure 1. Fundamental scenario for cryptography: Two entities, normally called Alice and Bob wish to share a secret which a third party, usually called Eve (who usually also harbours some malicious intent) is also interested in knowing.

exploring and devising applications that utilize non-classical world features, such as quantum entanglement and superposition. While elements of "quantum physics in action" have already been available to us on daily basis (e.g. barcode scanners, mobile phones), applications such as quantum computers, quantum cryptographic systems etc. that have much wider implications are now also getting deployed [Swiss Elections 2007; D-Wave on Wired, 2012]. The focus of this article is on quantum key distribution (QKD) – the most successful and well-known application of quantum cryptography, as of today. So much so that they are frequently used interchangeably, and this shall also be the case with this article.

Figure 1 illustrates the basic scheme of any cryptographic system. Two users Alice and Bob communicate privately (typically using a conveniently-available medium such as the Internet or telephone network) and an adversary Eve is interested in knowing the secret that they wish to share. Alice and Bob generally use a stream of bits called *keys* to protect their communication from being eavesdropped. While conventional schemes use the computational hardness offered by public-key cryptography (PKC) to prevent such eavesdropping, quantum cryptography relies on the *disturbances* – introduced by an eavesdropper – in the communication of the *secret key* between Alice and Bob.

These disturbances can be detected and quantified and indicate the ultimate level of security that can be obtained – this depends on the quantum cryptographic protocol employed. If the protocol runs successfully, Alice and Bob could use methods such as the one-time pad (OTP) for encrypting and decrypting the actual message (plain text) that they wish to share. A well-known property of OTP is its *information-theoretical immunity to cryptanalysis*. Also, to successfully eavesdrop in this scenario, Eve must crack the secret key in *real time*. This is, however, not the case with PKC: previously encrypted messages (that may still be of vital importance, e.g. military plans) could be cracked if sufficient computational resources are at disposal and/or faster mathematical algorithms are discovered.

With advances in quantum computing, the day is probably not far when this would actually happen!

II. A quantum-crypto protocol explained… without any quantum mechanics

I shall now endeavour to explain perhaps the most famous and widely-used QKD protocol. And without resorting to *any* quantum mechanics!

That sounds contradictory of course, because if it were possible to *classically* explain the innards of a quantum cryptographic protocol, then why do physicists delve into the *usually unusual* quantum theories at all? So I have to make some unrealistic assumptions that will be good enough for capturing the essence of how the protocol works, but (obviously) do not suffice to cover all the quantum-mechanical aspects. Assume that there exists *an unlimited supply of tiny magical balls* with the following characteristics:

C1. The state of a ball is encoded by a number N; for operation of the protocol, it is chosen to be amongst the first four positive integers, i.e. N = {1, 2, 3, 4}. Note that exactly two out of the four numbers in this set are:

- Odd (N = 1 and 3)
- Even (N = 2 and 4)
- More than 2 (N = 3 and 4)
- Less than 3 (N = *1* and 2)

We denote these four possibilities by keywords K = {OD, EV, M2, L3}, respectively. The encoding procedure *does not take a number as the input, instead, it takes a keyword* which is chosen in a random fashion from K.

C2. Uttering a keyword to a ball makes it randomly pick a suitable value of N that satisfies the constraint imposed by the keyword. E.g., if the uttered keyword is M2, both N=3 and N=4 are equally-likely candidates. The ball then encodes itself with the picked value. Table I below shows an example of 12 different balls encoded one after the other.

	1	2	3	4	5	6	7	8	9	10	11	12
Keyword used	**OD**	**M2**	**L3**	**EV**	**L3**	**M2**	**EV**	**EV**	**L3**	**EV**	**OD**	**M2**
Value of ***n***	**3**	**4**	**2**	**2**	**1**	**3**	**4**	**2**	**1**	**4**	**1**	**3**

Table 1. *Encoding of the magical balls. Balls randomly pick suitable values to encode themselves, e.g. the balls choose 2 and 4 when EV is invoked in slots 7 and 8, respectively. However, in slots 9 and 11, invoking two different keywords L3 and OD still results in the same value N=1 encoded.*

C3. Once encoded, a ball retains no memory of the keyword that was used to encode it. It does know N itself, but cannot be forced to reveal the value (even the person who encoded the balls cannot do it). However, the ball is programmed to truthfully and logically answer "Yes" or "No" (bit "1" and "0", respectively) to *one and only one* of the following two questions.

a. Is N odd (denoted by N_{OD})?
b. Is N less than 3 (denoted by N_{L3})?

Note, if a ball answers "Yes" to the above questions, it would have correspondingly answered "No" to

c. Is N even?
d. Is N more than 2?

In other words, the possibility of N being more than 2 *completely excludes* that of N being less than 3 and vice versa. Similarly, the possibility of oddness *complements* the possibility of evenness. So it suffices to say that asking *a* {b} is equivalent to *c* {d}.

Introducing some terminology that shall help as we proceed, we define the process of encoding the state as *preparation*, and the process of questioning a ball as *measurement*. Balls are thus prepared and measured in one of two conjugate pairs (even/odd or more/less) of possibilities; a possibility is technically called a *basis*.

C4. An attempt to pry open the ball triggers a self-demolition procedure and the ball vanishes in a puff of smoke! Also, after measuring (i.e. answering a single question) the ball falls silent: no amount of prodding or poking, pleading or praying can make it talk.

Steps of the protocol

Below are the steps that demonstrate how Alice and Bob can share a secret key using these magical balls:

S1. Alice encodes 'M' (ideally, M approaches infinity) magical balls in the safe environment of her lab/office. The balls are labelled sequentially and the encoding keyword for each ball is carefully recorded.

S2. She sends the labelled balls to Bob using normal postal services. Note that the postal services can be *unreliable*, *inefficient* or even *untrustworthy*!

S3. Bob receives 'W' balls (W≤M, as some could be lost on the way), assembles them sequentially, and *randomly* measures (asks questions *a* or *b* to) each ball. He carefully records the

corresponding answers. In the end, he obtains a bit sequence comprising of 0s and 1s ("Yes" = bit "1", "No" = bit "0").

S4. Bob contacts Alice using an authenticated public line, i.e. a communication channel open to passive eavesdropping. He tells her only of the questions he asked, but doesn't divulge the corresponding answers.

S5. For each ball, Alice compares her encoding keyword (i.e. the preparation basis) with the question (i.e. the measurement basis) asked by Bob. Whenever there is a match, *i.e. the preparation basis of Alice and the measurement basis of Bob coincide*; Alice can correctly predict Bob's answer. She tells Bob to keep all these (on an average, this would be W/2) instances and discard the rest. In this manner, she finally obtains a bit sequence comprising of 0s and 1s, which is identical to that of Bob. *This is the raw secret key*!

S6. Alice and Bob publicly compare a few randomly-chosen bits from their secret key. If these bits are indeed the same, then they assume the same holds for the rest of the secret key. Discarding the publicly-compared bits, Alice and Bob obtain the final secret key.

The steps described above are illustrated using Table II. The columns highlighted in green correspond to situations when Alice and Bob used the same basis. If Alice simply assigns EV = M2 = bit "0" and OD = L3 = bit "1", then after discarding instances where dissimilar basis were used, Alice and Bob are left with an identical bitstream.

	1	2	3	4	5	6	7	8	9	10	11	12
Keyword used	OD	M2	L3	EV	L3	M2	EV	EV	L3	EV	OD	M2
Value of N	3	4	2	2	1	3	4	2	1	4	1	3
Bob's question	N_{OD}	N_{OD}	N_{L3}	N_{OD}	N_{L3}	N_{L3}	N_{L3}	N_{OD}	N_{OD}	N_{L3}	N_{OD}	N_{L3}
Ball's answer	Yes	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No
Bitstream Bob	1	0	1	0	1	0	0	0	1	0	1	0

Table 2. Magical ball protocol in action

While Alice randomly encodes the balls, Bob randomly questions them. After discussing on a public channel, they are able to process their data such that they finally obtain the same secret key (bit sequence in green) with a high probability.

Tackling Eve

To make sure that the magical ball protocol (MBP) works in an adversary scenario as well, Table III shows the effects introduced by Eve as she tries to gain knowledge of the key. Eve is assumed to control the postal services, so the balls are essentially in her custody until they are delivered to Bob. Due to assumptions C1-C4, the most suitable methodology for Eve would be to interrogate the balls herself and depending on the answers, encode a new sequence of balls and send them to Bob.

	1	2	3	4	5	6	7	8	9	10	11	12
Keyword used	OD	M2	L3	EV	L3	M2	EV	EV	L3	EV	OD	M2
BitstreamAlice	1	1	1	0	1	0	0	0	1	0	1	0
Eve's question	N_{OD}	N_{L3}	N_{L3}	N_{L3}	N_{OD}	N_{OD}	N_{L3}	N_{L3}	N_{OD}	N_{OD}	n_{L3}	N_{OD}
Ball's answer	Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	No	Yes	Yes
Value of m	3	3	1	2	3	4	4	1	1	2	2	3
Bob's question	N_{OD}	N_{OD}	N_{L3}	N_{OD}	N_{L3}	N_{L3}	N_{L3}	N_{OD}	N_{OD}	N_{L3}	N_{OD}	N_{L3}
Ball's answer	Yes	Yes	Yes	No	No	No	No	Yes	Yes	Yes	No	No
BitstreamBob	1	1	1	0	0	0	0	1	1	1	0	0

Table 3. Foiling an eavesdropper

Eve intercepts Alice's communication, measures the balls herself and depending on the results, prepares a fresh sequence of balls

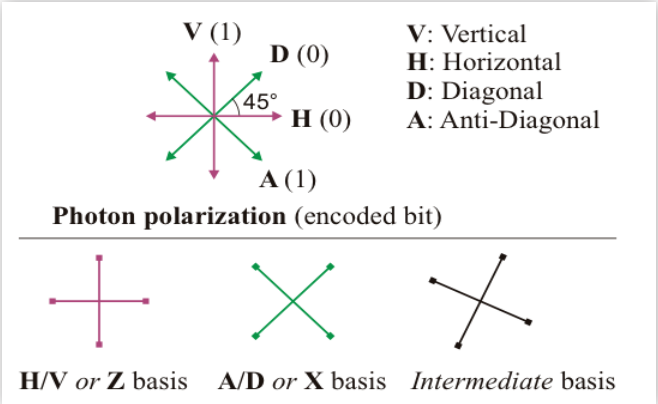


Figure 2. Bit encoding and polarization bases in typical QKD protocol

(with value ***m***) and sends them over to Bob. However, she inevitably makes a mistake, which Alice and Bob later discover for the cases where they used the same basis but don't get the same bit (shown in red). To distinguish between the legitimate users and Eve, the steps performed by the latter are shown in blue (first column).

In roughly half of all the 'W' instances, Alice and Bob used the same basis. In about half of *these* instances, Eve would've used a dissimilar basis. Thus, on average, Eve induces an error in one-fourth or 25% of all the instances, discovering which, Alice and Bob can decide to abort the protocol, discard the key and retry (or perhaps use a different communication link).

III. Quantum cryptography in practice

The result obtained in the last line of the previous section is identical to that achieved by the BB84 quantum cryptographic protocol that was proposed by Charles Bennett and Giles Brassard at a conference in Bangalore in 1984 [Bennett and Brassard, 1984]. The error rate, called quantum bit error rate (QBER), is the most important quantity relevant to the security of any QKD protocol. To elaborate, at the completion of a QKD protocol, if Alice and Bob incur an error rate above the threshold established by the theoretical security proof (of that protocol), then they discard the key. If the QBER is below threshold but non-zero, then Alice and Bob use privacy-amplification techniques [Deutsch *et al.*, 1996] to distill a secret key that reduces Eve's potential knowledge close to zero. An eavesdropper (bounded just by quantum mechanics) is thus destined to be discovered or thwarted and *even an unprecedented amount of computational power cannot help him/her to stay concealed*!

	Magical ball protocol	**Realistic BB84 protocol**
carriers of the encoded information	Tiny magical balls	Single photons
communication medium	Postal services	Quantum channel, e.g. optical fiber or atmospheric link
Bases for preparation and measurement	*Mathematical property*: Even/Odd (EV/OD), More/Less (M2/L3)	*Physical property* (*polarization*): Horizontal/Vertical (H/V), Diagonal/Anti-Diagonal (D/A)
physical method of realizing bits	Answers as either Yes or No	Binary system of optical detectors that either *click* or don't.
Other significant properties	Balls can be asked precisely one question, then fall silent	Single photons get destroyed once they are detected

Table 4. Analogy between MBP and realistic QKD protocol. Alice prepares photons in one of four possible polarizations (H,V,D,A) and sends them to Bob on a quantum channel. Bob measures the incoming photons using a system made from optical components and optoelectronic detectors. A successful detection event is registered when an impinging photon causes a click in the detector.

I shall now explain how a typical QKD protocol is implemented in real life. Table 4 strives to make a connection between the realistic implementation of BB84 with the magical ball protocol (MBP) described in the previous section.

A single photon is indeed the tiny magical ball granted to us by quantum physics. Conjugate bases for preparing and measuring photons obey the *Heisenberg Uncertainty Principle*: only one of the properties from a conjugate pair can be known with certainty. And simultaneously, the knowledge of the other (property) is completely randomized. For QKD, this translates to: *given only the outcome of a measurement, one cannot perfectly predict which basis had been applied*. This is indeed satisfied by MBP, however, Eve can – within the framework of quantum mechanics – resort to an *intermediate* basis. This carries no classical analogue and thus defines the realm where the functioning of MBP departs from that of a QKD protocol.

In practice, these bases can be easily constructed using the *polarization* property of photons [Gisin *et al*., 2002]. Figure 2 geometrically depicts four polarizations H, V, D and A along with the H/V basis and the D/A basis (usually denoted by X and Z, respectively) that follow the weird characteristics C1-C4 of the magical balls. In a real protocol, Alice starts by encoding classical bits randomly in either the X or Z basis; physically, it means that photons are randomly encoded in one of four possible polarizations. Note this differs from the MBP, the reasons for that shall become clear later. She then sends the photons over a quantum channel to Bob, who (independently) measures them in the X or Z basis.

The measurement outcomes are obtained using a specialized optical detection assembly made from single-photon detectors (SPDs). One such possible detector assembly inside Bob's device is shown in Fig. 3. The basis choice is implemented by a waveplate and polarizing beam splitter (PBS). A waveplate rotates the polarization of the input photon depending on the relative angle θ between its optic axis and the input polarization. A PBS on the other hand allows a single photon to be transmitted **{**reflected**}** if the photon is in a horizontal **{**vertical**}** polarization. The photon is then detected by D0 **{**D1**}**, where the latter numeral signifies the measurement outcome – the classical bit obtained.

Bob randomly chooses θ = 0° or 22.5° to apply the Z or X basis, respectively. Depending on the incoming state of polarization, a single photon gives a *click* in either of the two detectors. E.g. to communicate bit 0, Alice sends either a horizontally- or diagonally-polarized (H or D) photon. If the waveplate is set at θ = 0°, the H photon simply transmits through the PBS and gives a click in D0. But in this situation, a D photon would be transmitted or reflected by the PBS with an equal probability, so it could give a click in either D0 or D1. However, if the waveplate is set at θ = 22.5°, then the D photon is rotated and becomes an H photon, thus getting transmitted through the PBS and yielding a click in D0.

Some key features that apply to a majority of the currently-known QKD systems are summarized below:

R1. Periodic operation: The sender (usually Alice) emits optical signals, such as pulses of single photons, at well-defined times.

R2. Basis choice application: In theory, basis choice (either preparation or measurement) happens at a single instant of time. In practice however, this is usually implemented by application of an electronic voltage pulse which (obviously) has a finite *rise* and *fall* time.

R3. Dead time of the single-photon detectors: After a detection event, an SPD is rendered inactive for a short period of time. This is known as *dead time* and normally ranges from a few nanoseconds (ns) to a few microseconds (μs).

R4. Finite efficiency of detection and dark counts: Not every single-photon pulse that impinges on a SPD produces a click. Owing to technological limitations, typical single-photon detectors employed in QKD systems click only 10-15% of the times; this no. is described as the *detection efficiency* and is usually denoted by the Greek letter η. Moreover, a detector may click even in complete absence of optical input; this is known as a *dark count*.

The probability of obtaining a detection click from dark counts is typically smaller by a factor of 10^5 or more than from a single photon. Nonetheless, a legitimate bit 0 then requires *not just a click in D0, but also no-click in D1* (and vice-versa for bit 1).

Since SPDs operate at extremely low levels of light, they are activated only for the short duration when the periodic single-photon pulses from Alice are *expected* to arrive. The detector control, utilizing synchronization info (refer Fig. 3), electronically *opens a window* on both detectors at the time a single photon *may* impinge upon them. For the remaining time, the window is kept closed and the detectors are not much sensitive to stray light. The information about the measurement outcome in this time-window, i.e. '*which detector clicked*?' is subsequently passed onto the main board/computer.

The detector control & signal analysis circuit sends a voltage pulse to each of the detectors. Due to the finite response

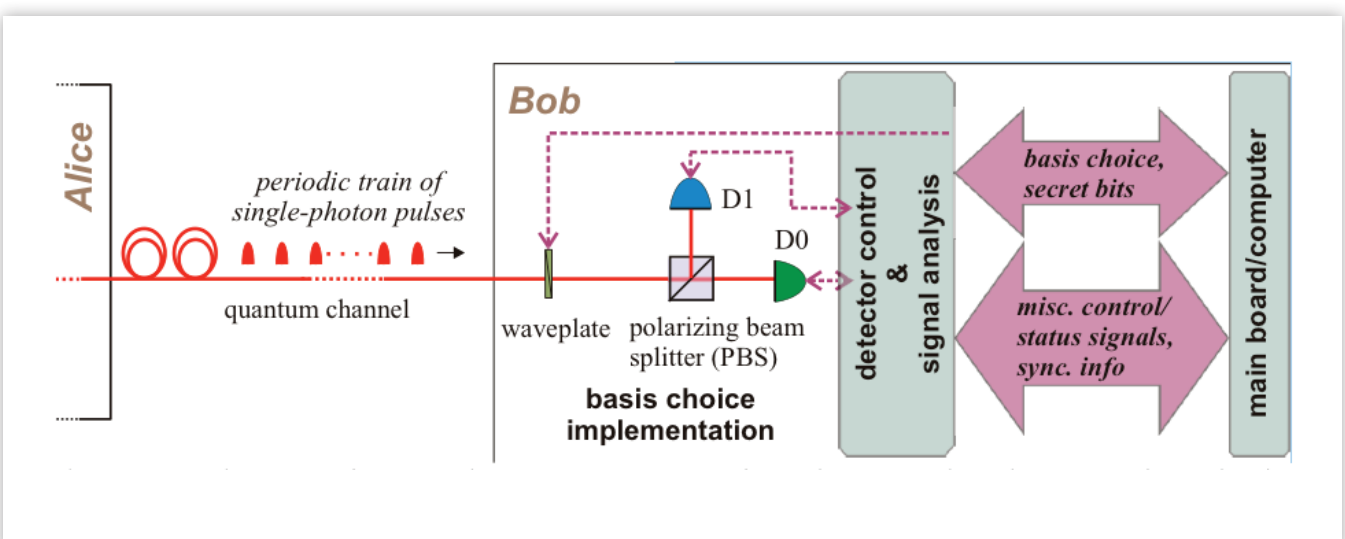


Figure 3. Bob Randomly Chooses schematic of a typical measurement setup for polarization based QKD

times, the corresponding efficiency curves rise and fall in a shorter interval. The peak value achieved depends on the optical & electrical properties of the detector material, the applied electronic voltage etc. In Fig. 4, a typical 'windowed mode' optical detection that considers R1-R4 is depicted.

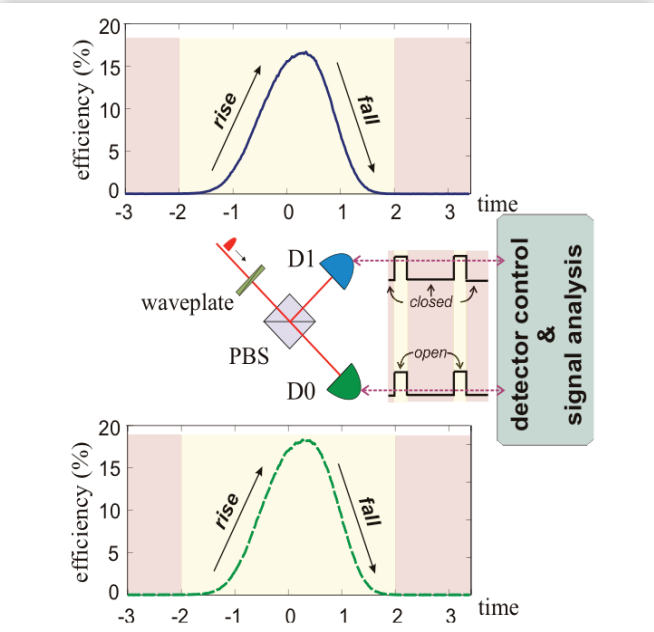


Figure 4. Detection efficiency as a function of time

IV. Quantum hacks & attacks based on timing

To obtain a tangible security primitive, e.g. the secret key bitstream, any physical QKD system thus *needs to make a transition from the quantum to classical domain*. Quantum physics may prove that a QKD protocol, operated under certain assumptions, is secure. But can practical implementations also certify that beyond doubt?

A quest for the answer to this question began roughly a decade ago and has led to some astonishing results [Leuchs, 2011]; see Fig. 5. Termed 'quantum hacking', this research field has witnessed many successful *proof-of-principle* attacks devised and performed on practical QKD systems. The attacks primarily show how an eavesdropper obtains partial or full info about the secret key without breaching the QBER threshold. It should be stressed that a majority of the eavesdropping strategies utilized differences between the security proof of the QKD protocol (a.k.a. the theoretical model) and the actual implementation. These differences mainly arise due to technical imperfections or deficiencies of the hardware, such as single-photon detectors.

The bar chart shows the growing no. of publications devoted to (experimentally) finding and exploiting limitations of practical QKD systems. Only papers from peer-reviewed journals have been considered, so this is by no means an exhaustive list. ID Quantique (*http://www.idquantique.com*) and MagiQ Technologies (*http://www.magiqtech.com*) were the first two companies to commercialize QKD.

The main topic of this article is quantum hacking activities that fall under the gamut of timing attacks. In the following, I shall discuss a few (published) works that share the common theme of exploiting *time* as a parameter or variable to compromise the security of QKD. I shall, while resorting to the minimum possible technical details, elaborate the following:

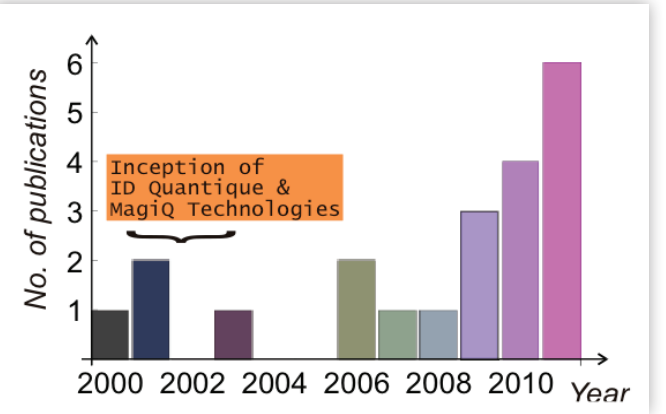


Figure 5. Rising popularity of quantum hacking

- Main exploitable vulnerability
- Principle of the attack
- Result and impact

General & specific countermeasures to prevent/detect these attacks and implications of quantum hacking on QKD will be discussed in the end.

A1. Effects of detector efficiency mismatch on security of quantum cryptosystems

For the correct working of a QKD protocol the measurement outcome should be random {deterministic} whenever the preparation basis is different than {equal to} the measurement basis. It entails here that to make sure this holds true for practical QKD, it is mandatory that the chances of detection by D0 and D1 are equally likely, i.e. *any discrimination between the detectors should not be possible*. However, realistically speaking, it is impossible to construct two identical detectors. Even if one manages to make them very similar, their respective detection windows – while individually being open during the arrival time of a photon – are still prone to shift relative to each other. This could happen, e.g. due to finite manufacturing tolerances, such as tiny optical path length differences or circuit length differences, which fluctuate because of temperature variations etc. and thus are very hard to control. In such a situation, the system exhibits a *detection efficiency mismatch* in time and this was the main topic of a paper [Makarov *et al*, 2006] by researchers from the Norwegian University of Science and Technology (NTNU) and St. Petersburg State Polytechnic University. Figure 6 illustrates the case of two detectors in Bob responding *differently* to single photons that impinge upon them at the *same instant of time*.

Furthermore, the researchers also theoretically proposed a special kind of *intercept-and-resend* strategy (see Table 4) to eavesdrop in such a scenario. Eve intercepts and measures the states sent by Alice on the quantum channel to Bob. Depending upon a measurement outcome, she resends a new state prepared in such a manner that it would produce a detection event in Bob, *if and only if* his measurement basis is opposite to Eve's preparation basis. In this way, Eve imposes faked detection events in Bob and hence, such an attack is referred to as the *faked-states attack*.

Although this paper was primarily a theoretical study, i.e. it did not demonstrate an attack on a QKD system as a whole, yet it is considered as a landmark paper because its main ideas have formed the core of countless eavesdropping strategies in last 5 years. In fact, two of the attacks that I shall discuss further on are based on detector efficiency mismatch (DEM) and two utilize the concept of faked states!

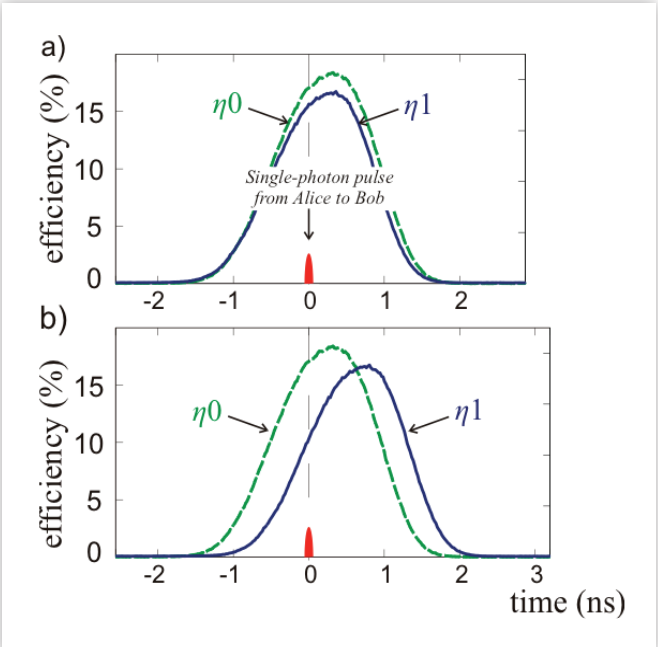


Figure 6. Temporal detector efficiency mismatch.
a) Apart from a minor difference in the peak value and overall shape, the efficiencies overlap each other quite well, or temporal mismatch is negligible. This would be alright for a practical QKD system. b) The mismatch is rather pronounced and could arise due to electrical differences, e.g. the length of the circuit wire to D0 may be slightly shorter than the one to D1, so the "window open" pulse dispatched by the detector control reaches D0 roughly 0.4 ns before D1. As a result, the single-photon pulse in b) sees a vastly different efficiency of detection.

A2. Breaking a quantum key distribution system through a timing side channel

It should be clear how critical it is for the detectors of a QKD system to appear indistinguishable to the outside world (especially Eve). Researchers from National University of Singapore in 2007 showed how timing information revealed during public discussion between Alice and Bob (S4 & S5 in the protocol, section 2) could allow an adversary to discern between the detectors and hence, eavesdrop without introducing perceptible errors.

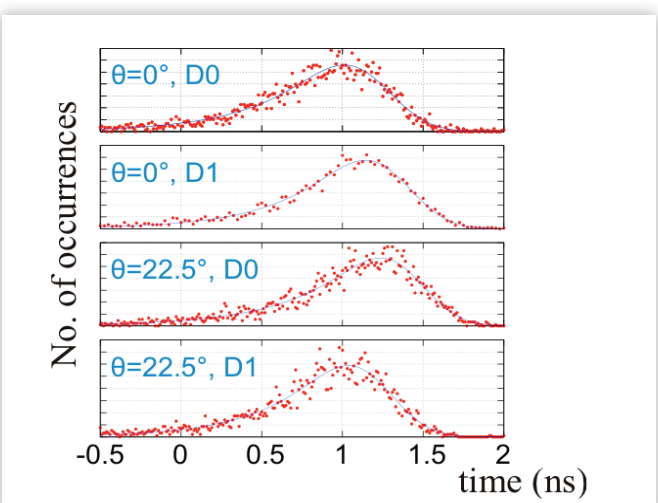


Figure 7. Timing side channel attack
Timing histograms collected from all four measurement possibilities (blue labels) show different centroid locations. At angle θ=22.5° (X basis) the separation between peaks of D0 and D1 is especially prominent; once the basis choice is revealed on the public channel, Eve has some chance to know if the obtained bit was 0 or 1. Figure reproduced with permission, ©2007 Optical Society of America.

As such it seems implausible to think of a correlation between the measurement outcomes and the publicly exchanged timing information (such as timestamps for synchronization). However, if the timing information is communicated (by Alice to Bob or vice-versa) with a high resolution, an eavesdropper could perhaps know the answer to *which detector clicked* just by carefully scrutinizing the timestamps! Figure 7 depicts a histogram of the optical detection timings from the compromised QKD system [Lamas-Linares and Kurtsiefer, 2007] and conveys the idea of this attack.

Technically, this may be due to the same reasons that cause the detector efficiency mismatch (explained in the previous attack). More qualitatively, this should be understood as *inadvertent* encoding of information about the secret key in undesired degrees of freedom. These degrees of freedom are formally called *side channels* and have been extensively explored in classical cryptography.

The info about the secret key that Eve could obtain with such an attack was also evaluated theoretically. Detection systems uncompensated by a mere 0.5 ns – an amount that can go easily unnoticed in usual experimental setups – could allow the eavesdropper an access ≥ 25% of the key! Thus, the loophole exposes an important issue which cannot be overlooked.

Experimental demonstration of time-shift attack against practical quantum-key-distribution systems

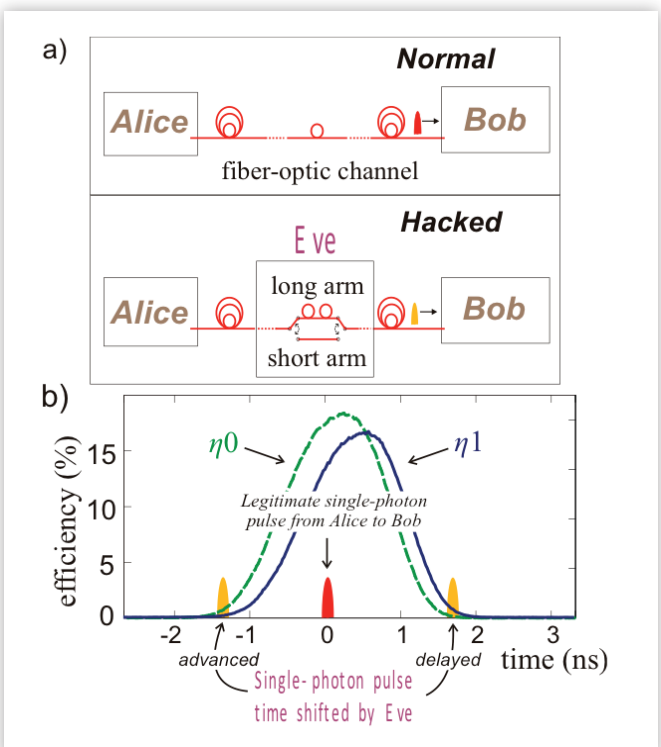


Figure 8. Time-shift attack
To eavesdrop, Eve simply shifts – advances or delays – the legitimate optical signals from Alice that are on the way to Bob. This could be achieved by using an assembly of optical switches and delay fibers as shown in the Hacked case.

Researchers from University of Toronto made an attempt to exploit the aforementioned DEM vulnerability by performing a time-shift attack [Zhao *et al*, 2008]. This was the first known (proof-of-principle demonstration of an) attack on commercial QKD, the system in this case was manufactured by the Swiss company ID Quantique. The idea behind the attack, illustrated

in Fig. 8, is quite simple. Eve cuts a section of the quantum channel and places an assembly of optical switches that reconnect Alice and Bob via two fibers, one *shorter* and other *longer* than the cut section.

From the viewpoint of *actual* execution, the time-shift attack (TSA) unfortunately suffered from numerous problems:

1) To apply the attack, the temporal mismatch needs to stay high, i.e. η0 and η1 peaks should be far apart, throughout the running of the protocol. As reported by the authors of this work, this occurred quite rarely (4% of all the instances).
2) The mismatch not only occurs infrequently, but is also probabilistic. The efficiency η0 could be left, right or on top of η1 but Eve *cannot predict* this at all. The QKD system also does not reveal any information that could facilitate Eve in this regard.
3) The magnitude of the highest mismatch is also not exceptional. Consequently, Eve has to displace the states to the extremities of the window for executing TSA. The overall detection rate seen by Bob (and Alice) would then be much lower than expected, this could raise alarm.

Thus, the efficacy of eavesdropping is considerably reduced. Nonetheless, the TSA was instrumental in bringing realistic QKD into limelight again and spurring a slew of quantum hacking activities.

After-gate attack on quantum cryptosystem

In 2010, researchers from NTNU, Max Planck Institute for the Science of Light (MPL) and Universidad de Guanajuato had experimentally demonstrated the *blinding* of single-photon detectors (SPDs) in two different commercial QKD systems [Lydersen *et al*, 2010]. This has perhaps been the most powerful and the best-performing hack on QKD so far. In 2011, they targeted yet another imperfection of these SPDs and based on the idea of faked states, they were able to remotely control the measurement outcome in Bob [Wiechers *et al*, 2011].

The limitation that they exposed was that an SPD can be *forced to click* when a sufficiently bright pulse (typically containing millions of photons) hits it in the *dead time*, i.e. outside the detection window (see Fig. 9a). The main principle of the attack is as follows:

1. Eve intercepts the legitimate quantum states from Alice and measures them. This could be done using an exact copy of the Bob device. Let us assume she uses basis X and detects bit 0.
2. She now prepares a bright pulse in the X basis and times its arrival in Bob such that:
   - If Bob chooses the same basis as Eve, all of the optical power from the bright pulse is diverted to D0.
   - If Bob chooses the other basis, then the power splits equally among both the detectors.

It turns out that for the SPDs in Bob, there is an optical power threshold for bright pulses above **{**below**}** which they would always **{**never**}** click! This situation is illustrated in Fig. 9b.

Due to impinging of bright pulses on a SPD, increment of dark counts (due to a phenomenon called *afterpulsing*) is encountered in practice. This could deter Eve, but using some clever strategies, the researchers were still able to simulate an attack that showed the possibility of eavesdropping with QBER < 11% at the operating conditions of the commercial system.

Device calibration impacts security of quantum key distribution

It has been assumed that Alice sends single-photon pulses and Bob opens a narrow detection window *at the appropriate time* to measure them. But how is such synchronization obtained in the first place? QKD systems usually run a *calibration routine* for this purpose: Alice and Bob exchange timing information to calibrate their devices, before any secret key exchange.

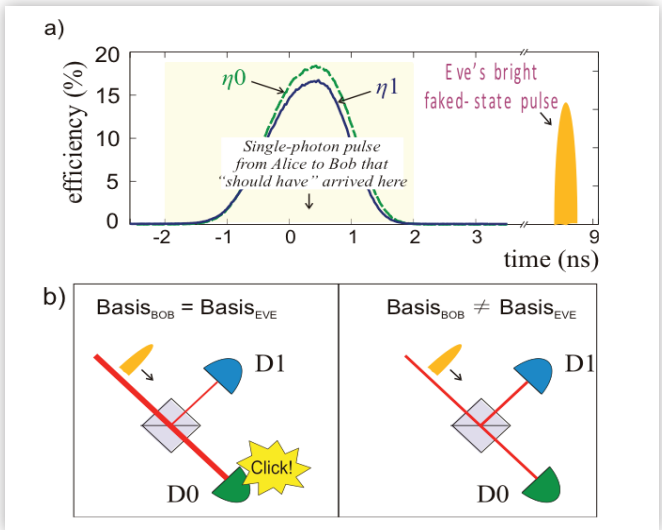


Figure 9. After-gate attack.
Eve launches bright pulses that arrive outside Bob's detection window, yet they make his detectors click according to her will.

The commercial QKD system from ID Quantique implements it (conceptually) as follows:

1. Alice prepares relatively-bright pulses (containing a few 1000s of photons) in the H polarization (bit 0 encoded in Z basis) and sends them to Bob.
2. Bob *always* measures in the X basis. Every photon in the pulse thus has an equal chance to yield a click in either D0 or D1.
3. Bob temporally scans the detection windows for D0 and D1 over the incoming pulses while monitoring the no. of clicks at each scan position.

By independently maximising the no. of clicks in each detector, Bob not only knows becomes synchronized to Alice, but also compensates for any detector efficiency mismatch (DEM). Figure 10a illustrates the optical scheme.

Researchers, primarily from MPL and NTNU, found a weakness in this calibration routine and experimentally demonstrated a way to hack it [Jain *et al*, 2011]; see Fig. 10b. Before the calibration pulses reach Bob, Eve manipulates them such that the photons in the first temporal half of each pulse are D- polarized (yielding clicks in D0) and in the latter half are A- polarized (yielding clicks in D1). An artificial displacement of the efficiencies is thereby *induced*, resulting in both large & deterministic DEM – already depicted in Fig. 6.

With this induced mismatch a *stronger* time-shift attack [Zhao *et al*, 2008] is possible, however, that would still not breach the security of the QKD system. Could eavesdropping by performing a faked-state attack (FSA) as explained in A1 be more successful? This work proved it so, and even under a very realistic operating regime of the QKD system. The QBER introduced by the FSA was simulated and by optimizing Eve's faked-state pulses (namely their arrival time into Bob and brightness), this QBER was minimized such that it did not exceed 8% and the expected detection rate of Bob was also maintained.

This attack does appear effective, but a specialized intercept-and-resend apparatus is needed to physically realize an eavesdropper and this is quite complex and expensive.

A3. Other attacks & countermeasures

The attacks described in **A1**-**A5** are of course not the only ones proposed/performed on practical QKD systems. Due to space constraints, I picked what all could be most easily tagged with 'timing attacks'. The interested reader is invited to explore contextually-similar works, e.g. the phase remap-

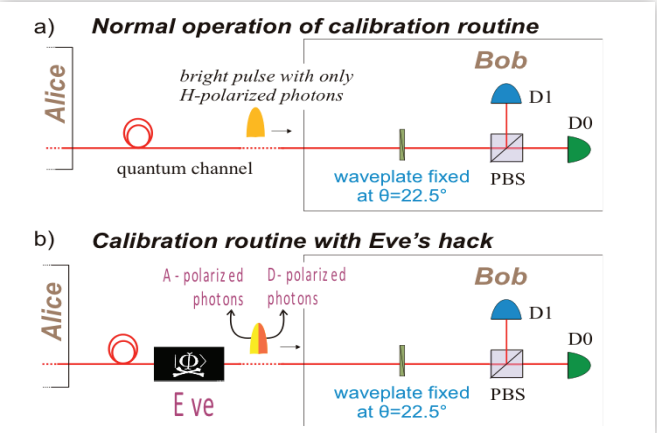


Figure X. Hacking QKD calibration to induce DEM and to pave path for a faked-state attack.
Eve spoofs the calibration pulses such that they still produce the expected no. of clicks in D0 and D1 while the detection windows are being scanned, but with a relative delay which induces a temporal DEM.

ping attack [Xu *et al*., 2010] and the dead time exploitation of single-photon detectors [Weier *et al*., 2011].

The detector efficiency mismatch (DEM) problem outlined in A1 has clearly enabled several timing-based attacks on QKD so far. It allowed eavesdropping via time-shift attack (A3), faked-state attack (A5), or through exposed timing side channels (A2). By carefully equalizing delays, randomizing basis choice and truncating precisions, DEM and associated vulnerabilities can be minimized and these attacks could be defeated. However, in general, a QKD system should still characterize timing-related information and subsequently account for any potential leakage of the secret key to Eve. Installation of watchdog detectors and optical isolators in Bob could thwart off bright pulse attacks (A4 and A5), however, they also decrease the legitimate throughput. Even more, such technical plugs & countermeasures that appear intuitively correct might (however) give rise to new and unthought-of loopholes. Hence, it is vital to carefully adapt them to the quantum protocol.

Another recent approach that advocates an altogether different paradigm of performing QKD is *device independence*: no knowledge of how the physical devices (of Alice and Bob) function is required! The only necessities are that quantum physics is correct and that Alice and Bob fully control the signals emitted and received by their devices [Acin *et al*., 2007]. Some new & exciting proposals in this direction have already claimed to get rid of most known detector flaws and side channels [Braunstein S *et al*., 2012; Lo H-K *et al*., 2012].

Conclusion

It is already speculated that public-key cryptography would be in great danger if and when quantum computers become a reality. In light of that, QKD is perhaps the best and most viable alternative.In this article, I've discussed a coterie of attacks against QKD systems. There is, however, a vast (and growing) literature – explored from both fundamental and technical viewpoints – on security concerns in practical QKD. The good news is that so far none of the attacks have been proven insurmountable. So, if we believe in laws of quantum physics, then we can also believe that QKD has a future. Nonetheless, it must be stress ed that the role played by physics guarantees *merely* a single link in the whole chain of security. In an excellent paper, very appropriately titled the black paper of quantum cryptography [Scarani and Kurtsiefer, 2009], the authors write:

"*We believe that this state of affairs cannot be simply dismissed with a 'there have been examples of bad design of the device'. At any stage of development, the devices were actually carefully designed; the security claims of the authors were accepted as valid by referees and colleagues. Neither now, nor at any later time, will one be able to guarantee that the devices in use are provably good. And it is certainly not a good idea to close one's eyes, invoke the laws of physics and dump on them a responsibility they cannot possibly bear.*"

At the beginning of section 4, the question "Will realistic systems that execute a quantum cryptographic protocol be able to certify its security claim beyond doubt?" was posed. The above quote provides a very suitable answer to that. So, a reasonable and acceptable level of security is definitely achievable, but like any other technology, practical QKD must also undergo a tight scrutiny to weed out vulnerabilities and patch loopholes.

Finally, I would like to remark that there are innumerable aspects of practical QKD that could not be covered by this article. These range from efforts on standardization of QKD to integrating it into public networks and even taking it into outer space using satellites. All in all, the take-home message is that quantum cryptography is quite a vibrant field with a tremendous potential for providing a secure communication technology in the future.

References

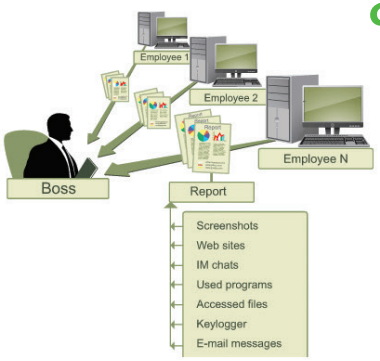
- Born M, 1969, *Physics in my generation*, Springer-Verlag, New York, 113
- EPR (Einstein A, Podolsky B and Rosen N), 1935, *Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?*, Physical Review 47, 777–780
- Swiss Elections 2007, *http://www.newscientist.com/article/dn12786-quantum-cryptography-to-protect-swiss-election.html*
- D-Wave on Wired, 2012, *D-Wave Defies World of Critics With 'First Quantum Cloud', http://www.wired.com/wiredenterprise/2012/02/dwave-quantum-cloud/all/1*
- Bennett C and Brassard G, 1984, *Quantum cryptography: public key distribution and coin tossing*, Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing, 175–179.
- Deutsch D, Ekert A, Jozsa R, Macchiavello C, Popescu S and Sanpera A, 1996, *Quantum Privacy Amplification and the Security of Quantum Cryptography over Noisy Channels*, Physical Review Letters 77, 2818
- Gisin N, Ribordy G, Tittel W and Zbinden H, 2002, *Quantum Cryptography*, Reviews of Modern Physics 74, 145.
- Leuchs G, 2011, *Quantum Key Distribution Coming of Age*, International Conference on Quantum Information, OSA Technical Digest.
- Makarov V, Anisimov A and Skaar J, 2006, *Effects of detector efficiency mismatch on security of quantum cryptosystems*, Physical Review A 74, 022313.
- Lamas-Linares A and Kurtsiefer C, 2007, *Breaking a quantum key distribution system through a timing side channel*, Optics Express 15, 9388.
- Zhao Y, Fung C-H F, Qi B, Chen C and Lo H-K, 2008, *Experimental demonstration of time-shift attack against practical quantum-key-distribution systems*, Physical Review A 78, 042333.
- Lydersen L,Wiechers C, Wittmann C, Elser D, Skaar J and Makarov V, 2010, *Hacking commercial quantum cryptography systems by tailored bright illumination*, Nature Photonics 4, 686.
- Wiechers C, Lydersen L, Wittmann C, Elser D, Skaar J, Marquardt Ch, Makarov V and Leuchs G, 2011, *After-gate attack on quantum cryptosystem*, New Journal of Physics 13, 013043.
- Jain N, Wittmann C, Lydersen L, Wiechers C, Elser D, Marquardt Ch, Makarov V and Leuchs G, 2011, *Device calibration impacts security of quantum key distribution*, Physical Review Letters 107, 110501.
- Xu F, Qi B and Lo H-K, 2010, *Experimental demonstration of phase-remapping attack in a practical quantum key distribution system*, New Journal of Physics 12,113026
- Weier H, Krauss H, Rau M, Fürst M, Nauerth S and Weinfurter H, 2011, *Quantum eavesdropping without interception: an attack exploiting the dead time of single-photon detectors*, New Journal of Physics 13, 073024.
- Acin A, Brunner N, Gisin N, Massar S, Pironio S and Scarani V, 2007, *Device-Independent Security of Quantum Cryptography against Collective Attacks*, Physical Review Letters 98, 230501.
- Braunstein S and Pirandola S, 2012, *Side-Channel-Free Quantum Key Distribution*, Physical Review Letters 108, 130502.
- Lo H-K, Curty M and Qi B, 2012, *Measurement-Device-Independent Quantum Key Distribution*, Physical Review Letters 108, 130503.
- Scarani V and Kurtsiefer C, 2009, *The black paper of quantum cryptography: real implementation problems, http://arXiv:quant-ph/ 0906.4547v1.*

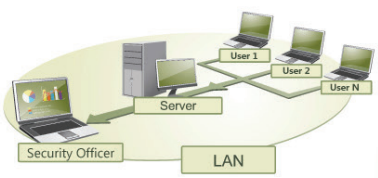
NITIN JAIN

obtained B.Tech in electrical engineering and M.Tech in communications & signal processing from IIT Bombay (India) in 2004. He worked in the software industry for a while, developing video codecs for handheld devices at NVIDIA Graphics Pvt. Ltd., Pune (India). But his love for physics, especially optics, was to eventually carry him back to the academic world. In 2007, notwithstanding the almost-antipodean change of longitudinal location (from 73° E to 114° W) and peak temperatures (+40° C to -40° C), he headed to the University of Calgary (Canada) to do an MSc in physics with main focus on quantum information science & technology. Since 2010, he has been pursuing a doctorate at the Max Planck Institute for the Science of Light, Erlangen (Germany). His thesis work is related to the hacking of practical quantum key distribution systems.

STAFFCOP

PC monitoring, Corporate Security and Data Loss Prevention Software





StaffCop Standard allows you to monitor all activities on company computers and prevent the unauthorized distribution of sensitive corporate information.

StaffCop will help you:

To locate possible data loss channels and prevent loss
To gain insight into how your employees spend their work time
To increase company and departmentals efficiency

You need StaffCop to:

Gather work time efficiency statistics
Easily control your employees in real-time mode
Improve discipline and motivation of your employees

Who needs StaffCop:

CEO/CTO
Corporate Security Manager
HR Manager
System Administrator

Main Features of StaffCop:

Screenshot recording
Application monitoring
E-mail monitoring
Web site monitoring
Chats/IM activity recording
USB device monitoring
Clipboard monitoring
Social Networks Monitoring
Search Term Tracking
File and Folder tracking
Keystroke recording
System Event Monitoring
Whitelists and Blacklists
PC activities reporting
Stealth installation/monitoring
Strong security
Alert notifications
Remote Install / Uninstall

More Information, Demo Versions, Videos and Technical Guides -

www.STAFFCOP.com

Phone: +1-707 -7098405
Skype: staffcop.com
Email: sales@staffcop.com, paul@atompark.com

Microsoft CERTIFIED Partner

Do You Want to Become a Cyber Security Expert?

OR ADVANCE YOUR IT SECURITY CAREER?

- Cyber Security has one of the largest market shares in IT
- Government & Compliance Regulations are more and more enforced
- Gartner Group predicts unprecedented growth and need in Cyber Security
- Skilled Cyber Security Experts are in ever more demand

THE CYBER 51 EXPERT COACHING FORUM

- Individual 1-on-1 Mentoring on Ethical Hacking, Penetration Testing and IT Security
- Networking with other community members and moderators
- Access to a wealth of tools and information not found on public domain
- Permanent Job & Contract offers, Webinars and much more!

YOUR BENEFITS

- Become an Ethical Hacker / Penetration Tester with 1-on-1 mentoring
- Learn at your own pace at a fraction of the cost of regular courses

CYBER 51 COACHING FORUM

CYBER SECURITY FORUM

CONTENT:

1. General Topics
2. Service Assessment
3. Ethical Hacking
4. Cyber Threats
5. Mitigating Cyber Threats
6. Penetration Testing

CYBER 51 INSTRUCTORS

OUR CERTIFICATION LEVELS:

- Certified Ethical Hacker (C|EH)
- Forensic Investigator (C|HFI)
- Certified Security Analyst (ECSA)
- Licensed Penetration Tester (C|LPT)
- Network Security Admin (ENSA)
- ISC Consortium (CISSP)

FEATURES

ADDITIONAL FEATURES:

- 1-on-1 Coaching
- Trainers with Years of Experience
- Wealth of Tools
- Webinars
- Networking with other members
- Contract & Perm. Job Opportunities

WHY CYBER 51?

- Learn whenever you want to
- Dedicated 1-on-1 Coaching
- Information you will not find on public boards
- All Mentors work as Senior Security Consultants
- Frequent updates
- Great Value for money

CYBER 51
A PURE CYBER SECURITY NETWORK

CONTACT US TODAY

CYBER 51 LIMITED, 176 THE FAIRWAY, SOUTH RUISLIP, HA4 0SH, MIDDLESEX, UNITED KINGDOM

EMAIL: INFO@CYBER51.CO.UK

WEB: WWW.CYBER51.CO.UK

ATOLA BANDURA: SUPERFAST IMAGER, WIPER, AND TESTER

An interview with Vitaliy Mokosiy, the Atola Bandura project manager and head developer

VITALIY MOKOSIY

is the Atola Bandura project manager and architect. He has been working in Atola Technology as an expert in software development of HDD tools for data recovery and forensics since 2008. Vitaliy is also known as the project manager of Atola Disk Recycler and as the lead software developer of Atola Insight. His success in all projects is a mix of more than nine years of .NET and Java development experience and team management capabilities.
LinkedIn profile: http://www.linkedin.com/in/vitaliymokosiy

Vitaliy Mokosiy is the lead Atola Bandura developer. He is an expert in the development of HDD tools for data recovery and forensics. Vitaliy kindly agreed to share some information about Atola Bandura with dearly beloved readers: its history, development, opportunities, and advantages in discovering the tool's value.

What is the history of Atola Bandura? What made you decide on its development? Who are the main developers?
As for me, the development process of Atola Bandura itself was extremely exciting. During the process, I encountered lots of interesting things, starting from idea conceptualization to our first market delivery.

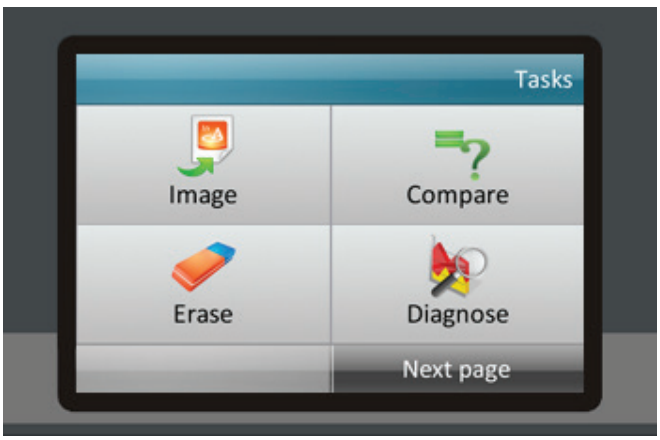
The great success of Atola Insight project lead by Dmitry Postrigan has brought us to designing a new and innovative system. We expected Bandura to possess the following features:

- Full-color, easy to use touch screen user interface (UI)
- High-quality duplication of damaged hard disks without connection to a PC (stand-alone mode)
- Maximum possible imaging and wiping speed

Thus, we strived to make the process of testing, duplicating, or erasing of any HDD very easy and straight-forward for our users. As a result, Atola Bandura was specifically designed as a high-speed and easy-to-use imager. It's like a Swiss army knife that includes many additional features (disk diagnostics, checksum calculation, disk comparison, bad sector repair, HPA/DCO, etc.).

How much time did you spend on designing? Did you have any specific difficulties?
In general, the period from idea conceptualization to delivery took about eighteen months. Our team that developed the first edition of Atola Bandura consisted of two hardware engineers, two software developers, and two quality assurance engineers.

The biggest problem we had was with the touch screen. We had to redesign it twice to improve the quality of the image and its responsiveness.

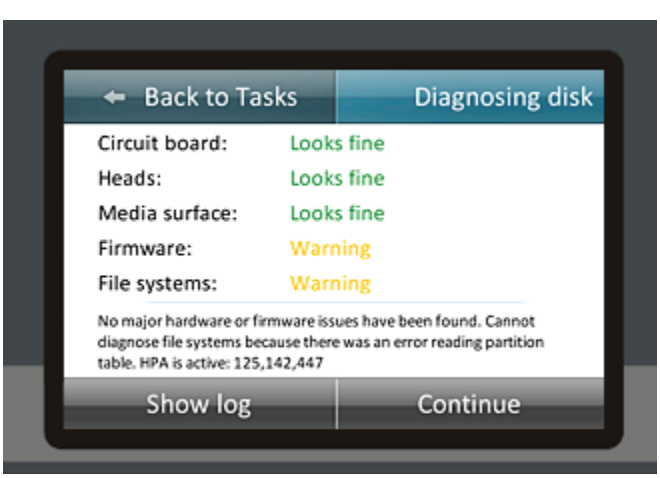


A large amount of time was spent on engineering and implementing the user-friendly touch screen interface. The concept was to make it extremely simple and easy for any Bandura user from the very beginning. As a result, one has to tap only once or twice to access any tool's function. For example, to provide disk diagnostics, all you have to do is connect the hard drive to the unit and tap "Diagnose" in the Main Menu. In a few minutes, Atola Bandura displays on the screen all possible problems within the recommendation report based on the analyses of the circuit board, head stack, media surface, firmware, SMART, and the availability of file systems.

When exactly can Atola Bandura be used?

Atola Bandura is a multipurpose high-speed tool that can be used in different business areas and situations. It can be applied in the following cases:

- **Case 1.** *Recovering data from damaged hard drives*. The smart multipass *Image* function allows extracting data from damaged hard drives by building an incremental copy in several passes.
- **Case 2.** *As an addition to bigger data recovery systems*. For example, if you're using the Atola Insight system as a data recovery specialist, you can use Bandura to take over several time-consuming functions from Atola Insight:
  - Fast disk imaging
  - Securely wiping two HDDs simultaneously
  - Comparing two disks through byte-to



- **Case 3.** *Tool for forensics business*. For forensics specialists, we have added the secure wiping of disks through different algorithms like NIST 800-88, DoD 5220.22-M, Security Erase. Features like change of HPA/DCO restrictions and checksum calculation (MD5, SHA1-512) are also available.
- **Case 4.** *Fast disk duplication*. Atola Bandura deals with any hard disk drive with or without bad sectors on a maximum possible speed rate. Also, there is an option to copy only sectors that contain data, thus saving even more time.

What are the specific characteristics of the product?

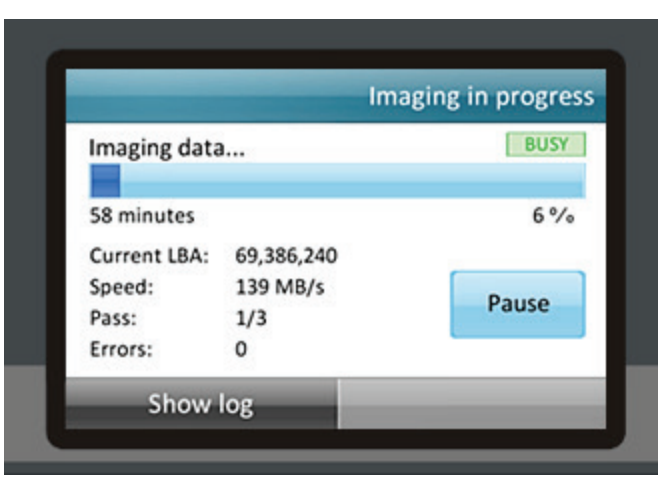
The main feature of Atola Bandura is its fantastic simplicity. To access any of its functions, the user must take the following steps:

1. Connect one or two disks.
2. Select any operation on the touch screen. Most functions start immediately, while others require minimal tuning on an additional screen.
3. Now the selected operation is starting. During its progress, you have access to statistics and the log.
4. When the process is finished, the result is shown on the screen. Additionally, reports are automatically saved to the external USB flash.

You can see the excellent work of At ola Bandura's imaging on YouTube, *Disk duplication with Atola Bandura,* at *http://www.youtube.com/watch?v=K3wM0U9-3qI* (or search *Atola Bandura* on YouTube).

What exactly should any user know regarding this tool? Are there some specific technical features?

Bandura provides quick and efficient imaging of damaged hard drives. The maximum speed rate of imaging is 256 MB/s. It is only limited by the hard disk's internal transfer rate. Also, it is very important to point out that you can stop the imaging process at any time, and you may resume it later.



I would like to emphasize the following features: a colored 3.3-inch screen, erasing speed up to 280 MB/s, write protection for source port, autosaving of all results and steps during the process to the USB flash, firmware updates through the same USB flash, etc.

By the way, all Bandura firmware updates are totally free.

In what place in the data recovery market does Atola Bandura belong?
Atola Bandura is the best tool in its market segment from my point of view. It offers the highest level of efficiency and performance rate with a minimum amount of actions. Furthermore, the user is provided with not only disk function reliability, but also aesthetic appeal with its simple UI design.

Have you received any client feedback? What do users say about Atola Bandura?
I can share with you one of the feedbacks I received:

> *Yes, that's exactly what I was asking, and it would be great to see the Bandura with this feature, just something from the actual device that has performed the operation to back up any documentation that we issue to the client.*
> *Thanks, Vitaliy, I'm looking forward to the next firmware update. The last one was a major improvement to the Bandura, and my unit is working very well . . . Thank you and to all the guys at Atola for your hard work.*
>
> **Steve Cook,**
> **director at East Anglian Data Recovery Services**

I would like to add that it is always great to hear any feedback from clients. Positive feedbacks motivate our team, while negative ones allow us to react on the clients' complaints to make Atola Bandura more robust.

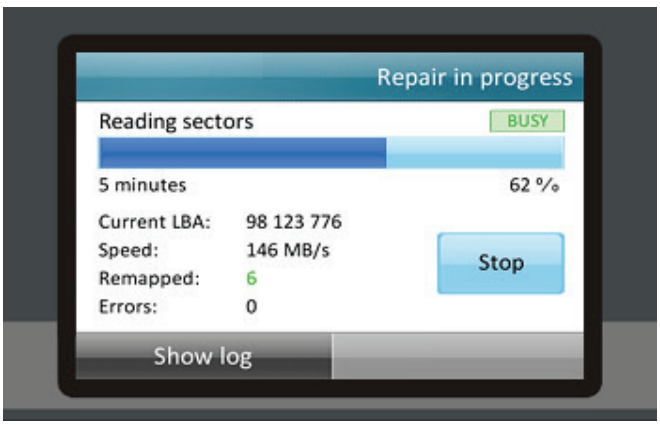
How much time and effort do you need to prepare Atola Bandura for operation?
To work with Atola Bandura, you actually don't need any special effort. To get the tool started, you have to connect it to a power cable and switch on the *Unit Power* button. After a minute, the Main Menu appears on the screen.

I would like to draw your attention to the fact that you do not need any special skills to work with Atola Bandura in a professional manner. You don't need to waste your time learning how to use it – it's that easy to use from the get-go.

What are the unique features of the tool?
Atola Bandura is a combination of convenience, fantastic speed rate, and efficiency in working with damaged hard disk drives. Emphasized features at a glance are as follows:

- Convenient touch screen UI
- Great results in working with damaged drives
- Superfast duplicator, wiper, and tester
- Includes many additional helpful functions comparable to that of a Swiss army knife

Which IT areas are indispensable to Bandura? Who are your potential users?
The biggest interest regarding Atola Bandura is shown in the data recovery and forensics areas. Also, our product can be used in nearly all IT business spheres (e.g., computer shops providing data recovery services for their customers).



Certainly, the majority of Atola Bandura users admire its imaging efficiency. But at the same time, users like Atola Bandura because it does not need a connection to a PC. Also, great importance is placed on the multifunctional characteristics of the device.

In my opinion, all potential users should not only estimate the value of various different functions, but also take into account the high efficiency of the tool as well as the lack of need to learn how to use it. Anybody can use Atola Bandura successfully.

What are your expectations for Atola Bandura in the future?
I can say without any doubt that Atola Bandura is the perfect solution for the data recovery market. And I strongly believe in its fantastic future because of its unique features.

The Atola Technology team made all efforts to create a tool that meets all needs of highly professional specialists as well as start-up businesses. Moreover, the feedback we receive from our customers reaffirms its high-level efficiency and functionality. Atola Bandura was designed to make each experience with it a pleasure. I'm proud to be part of such a great outcome!

You can find more information about Atola Bandura at *atola.com/products/bandura/*.

Reimagined for Small Business

COMODO ENDPOINT SECURITY MANAGER 2.0

Centrally manages the deployment of award-winning Comodo Internet Security software to protect the desktops and servers your business relies on against internal and external threats.

COMODO®

www.comodo.com

Develop for the Next Big Platform!

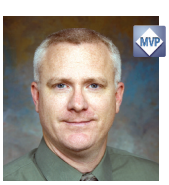
Attend the Windows Phone Developer Conference and get the best developer training!

WP DevCon

The Windows Phone Developer Conference

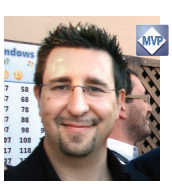
October 22-24, 2012
Hyatt Regency
Burlingame, CA
www.WPDevCon.net

Learn from the top experts at the Windows Phone Developer Conference, including 12 Microsoft MVPs!


Darrin Bishop


Michael Cummings


Nick Landry


Jose Luis Latorre


Chris Love


Colin Melia


Walt Ritscher


Lino Tadros


Kelly White


Shawn Wildermuth


Chris Williams

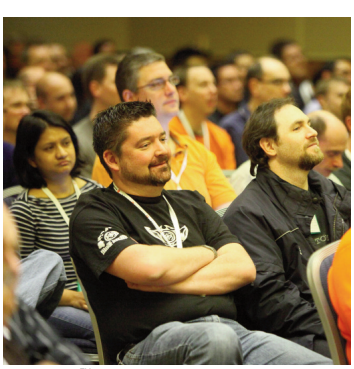

Chris Woodruff

50+ Classes and Workshops

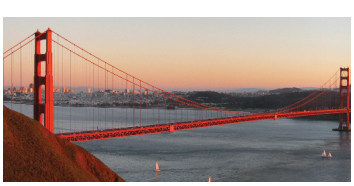
focus on a variety of important topics:

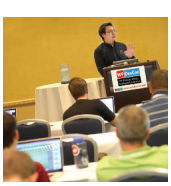
- Design implementation
- Location intelligence services
- Rich data visualization and implementation
- Cloud-based mobile solutions
- Development leveraging HTML5
- User experience
- Application design
- HTTP protocol
- Building reusable components
- Microsoft push notification service
- Creating custom animation
- and many more!

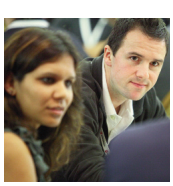
Visit WPDevCon.net for a full list of speakers, bios, classes, workshops, and special events!



Learn, network, and seize the opportunities that Windows Phone represents.







Register Early for the biggest discounts! at www.WPDevCon.net

Produced by BZ Media SD Times @WPDevCon