Reasoning about the Trade-off between Security and Performance

Boris Köpf
IMDEA Software Institute

Joint work with Goran Doychev
S4CIP, 25.05.2016
Performance is Paramount

CPUs
Compilers
Virtual machines
Networks

minimize

time
space
energy

... consumption

...
Performance-enhancing Techniques…

- Caching
- Concurrency
- Speculative execution
- Compression
- …
...and their Impact on Security

- Caching
- Concurrency
- Speculative execution
- Compression
- ...

Reduce resource consumption on average

Side-channel Attacks

Cache-timing attacks on AES

Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds

On the Power of Simple Branch Prediction Analysis

Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations

Exploit variations in resource consumption
Example: Control-flow Side Channel

Input: key $k$, ciphertext $c$

$x := 1$

for $i = 0$ to $|k| - 1$

$x := x^2 \mod n$

if $k[i] == 1$ then

$x := x \cdot c \mod n$

return $x$

• First attack: 1996 (Kocher)

• Practical, remote attack: 2003 (Boneh & Brumley)
Example: Cache Side Channel

- First attacks: 2005 (Bernstein/Shamir)
- Increasingly effective attacks: 2009-2015 (Ristenpart/…/Yarom)
Example: Traffic Side Channel

- Has been exploited for identifying language (Wright ’07), phrases (Wright ’08), speakers (Backes ’09), conversations (White ’11)
Closing Side Channels, Radically

- CPUs
- Compilers
- Virtual machines
- Networks
- ...

flatten
minimize
time
space
energy
consumption

Dramatic performance penalties
Reasoning about the trade-off between security and performance

A. How to quantify security?
B. How to quantify performance?
C. How to identify the sweet spot?

This Talk: A.+B.+C. by example, using information theory and game theory
Part I: The Case of Input Blinding

- Widely deployed countermeasure for timing attacks against public-key crypto (OpenSSL, PolarSSL, …)

\[ \text{Known Input} \rightarrow \text{RSA} \rightarrow \text{Timing Measurement} \]

![Diagram](image)

= randomize ciphertext before decryption, de-randomize plaintext after decryption
Information-theoretic Analysis of Blinding
[K. & Dürmuth, CSF ’09]

- We consider systems with deterministic timing behavior
  \[ t : S \times M \to O \]
  - Secret key (fixed)
  - Blinded message (i.i.d. chosen)
  - Timing observations

- In an attack, the adversary makes \( n \) timing observations
  \[ o_1 = t(s,m_1), \ldots, o_n = t(s,m_n) \]

\[ \leq \log |O|^n \text{ bits} \qquad \leq \log(n + 1)^{|O|} \text{ bits} \]

Good news or bad news?
An Adjustable Countermeasure

Blinding + "Bucketing" = Security guarantees

Large $|O|=3$

Performance overhead

Trade-off security vs. performance
The Countermeasure Configuration Game

- **Defender**
  - chooses defense strategy $d$ (bucketing, key length)
  - has utility:

- **Adversary**
  - chooses attack strategy $r : d \mapsto (n \text{ online steps}, m \text{ offline steps})$
  - has utility:

Stackelberg game: defender chooses before adversary

Challenge: express “security” in terms of the strategies
Security against Combined Attacks

Cryptographic hardness assumption in terms of unpredictability entropy [Hsiao ’07, Maurer ’11]

For all Adv: \( P[\text{Adv recovers key after } m \text{ computation steps}] \leq p(m, k) \)

implies

For all Adv: \( P[\text{Adv recovers key after } n \text{ timing observations and } m \text{ computation steps}] \leq p(m, k)(n + 1)^{|Ω|} \)
Unpredictability Entropy of Real Problems

• Only few facts known about unpredictability entropy
  • RSA (i.e. factoring) has low unpredictability entropy [Heninger ’09]
  • ElGamal (i.e. discrete log) seems safe [Maurer ’11]

• Theorem [Shoup ‘97]: Let $G$ be a cyclic group of prime order $q$ with generator $g$. Given $g^x$, the probability $p$ of computing $x$ using $m$ group operations is upper-bounded by

$$p \leq \frac{m^2}{q}$$

Unpredictability entropy of computing discrete logs for generic adversaries.
Bounds for Combined Algebraic/Timing Attacks
[Doychev & K., CSF ’15]

- **Theorem:** For an adversary that can make $m$ group operations and $n$ timing measurements of a blinded channel, the probability $p$ of successfully computing discrete logarithms is bounded by

$$p \leq \frac{m^2(n + 1)^{b-1}}{q}$$

Key length and bucketing
“Online” and “offline” steps
The Countermeasure Configuration Game

- **Defender**
  - chooses defense strategy $d$ (bucketing, key length)
  - has utility: $U_D(d, r(d)) = -\text{average execution time}$
    such that $p(d, r(d)) \leq p_{\text{max}}$

- **Adversary**
  - chooses attack strategy $r : d \mapsto (n \text{ online steps}, m \text{ offline steps})$
  - has utility: $U_A(d, r(d)) = p(d, r(d))$
    such that $n t_{\text{on}} + m t_{\text{off}} \leq \Delta$

We compute the optimal countermeasure configuration as an equilibrium in this game.
Case Study

- We analyze the ElGamal implementation of libgcrypt 1.6.1 (used, e.g., in GnuPG)
- Our goal is to find the implementation with best performance, for a given degree of security $p_{max}$
  - we compare implementations with different key lengths and numbers of buckets
  - we measure performance by counting the number of executed CPU instructions (and ignore influence of caches)
Results: Key lengths

![Bar chart showing the average cost in number of CPU instructions for varying modulus sizes.]

- **1 bucket**
- **2 buckets**

Reference modulus sizes: 1024, 1536, 2048, 2560, 3072

- **100 accesses per second**: A comparison with 365 days, 100 accesses per second.

**Varying the Number of Buckets:**

- Single bucket vs. two buckets for varying deployment time.

**Using a Safe Prime Modulus:**

- For fixed deployment time and varying bit size,
- The defender's preference may shift from a constant to a non-constant time implementation.

**Varying the Key Deployment Time:**

- Varying deployment time has a similar effect as varying the access rate.

- For bigger modulus sizes, we obtain optimal defense to use a constant-time implementation, i.e.,

\[ \text{duration (in days)} \times 10^8 \]

- An alternative approach for key generation makes sure that the group approach for key generation makes sure that the group

**100 hs/sec**

365 days
**Results: Deployment time**

In the following we choose the parameters to reflect the true entropy of the random number generator. We keep the parameters constant and vary only the deployment time of the key. The true entropy of the random number generator is expected to increase by a factor of 1.15, which is the increase observed in the data.

### Varying the Number of Buckets:

- For 1 bucket, the true entropy of the random number generator is expected to increase by a factor of 1.15.
- For 2 buckets, the true entropy of the random number generator is expected to increase by a factor of 1.15.

### Using a Safe Prime Modulus:

- Safe primes provide more security guarantees than default libgcrypt keys.
- Safe primes are more expensive to compute in terms of Theorem 3.1.

### Varying the Key Deployment Time:

- For example, if the deployment time is decreased, the adversary has more time to collect timing observations.
- For example, if the deployment time is increased, the adversary has less time to collect timing observations.

### Example Use Cases:

- The results are given for Figure 2: Varying the modulus sizes for default libgcrypt.
- The results are given for Figure 3: Varying access rate.
- The results are given for Figure 4: Varying the number of buckets.

### Summary:

- The constant-time implementation is preferred for key generation and key usage.
- The non-constant-time implementation needs to increase the corresponding modulus size to compensate for the information loss.
- The non-constant-time implementation needs to collect timing observations; thus, a defender deploying a non-constant-time implementation cannot decrease the key rate.
- The average cost (in number of CPU instructions) for optimum to shift from the non-constant time implementation enough for the modulus size to compensate for this information loss.
- The optimum at a non-constant-time implementation with two bucketing as modulus sizes.
- The corresponding optimal modulus sizes.

### Table:

<table>
<thead>
<tr>
<th>Bit Length</th>
<th>Average Cost (in number of CPU instructions) for optimum to shift from the non-constant time implementation enough for the modulus size to compensate for this information loss.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>1.05 × 10^8</td>
</tr>
<tr>
<td>1536</td>
<td>1.05 × 10^8</td>
</tr>
<tr>
<td>2048</td>
<td>1.05 × 10^8</td>
</tr>
<tr>
<td>2560</td>
<td>1.05 × 10^8</td>
</tr>
<tr>
<td>3072</td>
<td>1.05 × 10^8</td>
</tr>
</tbody>
</table>

### Key Generation:

- The group size of the group used for key generation is guaranteed to have a bit length of 100 bits.
- For example, the pycrypto library uses Algorithm 4.86.

### Conclusion:

- For bigger modulus sizes, we obtain the optimum at a non-constant-time implementation with two example use cases. We set the deployment time has a similar effect of varying the bit-length, safe primes provide more security guarantees than default libgcrypt keys. The reason is that, for a fixed bit-length, safe primes are more expensive to compute in terms of Theorem 3.1.

### References:

- Varying access rate:
  - Modulus sizes vary.
  - Results are given for Fig. 3: Average cost (in number of CPU instructions) for optimum to shift from the non-constant time implementation enough for the modulus size to compensate for this information loss.

### Figures:

- Figure 2: Varying the modulus sizes for default libgcrypt.
- Figure 3: Varying access rate.
- Figure 4: Varying the number of buckets.

### Notes:

- The defender has to increase the corresponding modulus size in order to ensure that the desired security level is met; i.e., the defender needs to increase the corresponding modulus size in order to ensure that the desired security level is met.

### Key Size:

- For example, the pycrypto library uses Algorithm 4.86.

### Algorithm:

- The group size of the group used for key generation is guaranteed to have a bit length of 100 bits.
- For example, the pycrypto library uses Algorithm 4.86.
Results: Access Rate

3076 Bit keys
365 days
Conclusions

• Presented an approach to systematically trade security for performance
  • derived formal bounds on success probability in terms of countermeasure configuration and numbers of online/offline steps
  • identified and computed optimal configuration as equilibrium in a game

• Made the case for a fast but (slightly) leaky implementation over a constant-time implementation!
  • very specific scenario (ElGamal signatures, input blinding, etc.)