Designing the API for a Cryptographic Library
A Misuse-Resistant Application Programming Interface

Christian Forler  Stefan Lucks  Jakob Wenzel

Bauhaus-Universität Weimar

Ada-Europe 2012

June 12, 2012
Outline

1 Introduction

2 Common Flaws in Cryptographic Applications
   - Buffer Overflows
   - Nonce Reuse
   - Plaintext Leaking

3 Summary
The Gap Between Theory and Practice

(Academic) cryptographers $\leftrightarrow$ (Industrial) engineers

work on

- technically cool systems
- provably secure
- practical systems
- hopefully secure
The Gap Between Theory and Practice

(Academic) cryptographers $\leftrightarrow$ (Industrial) engineers

work on
- technically cool systems
- provably secure
- practical systems
- hopefully secure

know
- how to design good cryptography
- why cryptosystems are secure or why not
- how to implement useful systems
The Gap Between Theory and Practice

(Academic) cryptographers ↔ (Industrial) engineers

work on
- technically cool systems
- provably secure
- practical systems
- hopefully secure

know
- how to design good cryptography
- why cryptosystems are secure or why not
- how to implement useful systems

when things go wrong

- Why didn’t THEY listen to us?
- Why didn’t THEY tell us?
Bridging the gap between theory and practice

Rise awareness of cryptographic misuse issues

Introduce our cryptographic library (LibAdaCrypt) (http://github.com/cforler/Ada-Crypto-Library)

Collecting design features to improve this Library
Common Flaws In Cryptographic Applications

Top Three Flaws In Cryptographic Applications

1. Buffer Overflows
2. Nonce Reuse
3. Plaintext Leaking
Buffer Overflows

Overrun boundary of a buffer and overwrites adjoining memory
Buffer Overflows

Overrun boundary of a buffer and overwrites adjoining memory

Countermeasure

Use of bounds checking programming language like Ada
A: “Have you any problems with encryption?”
B: “No, we are fine. We are using AES!”
A: “Well . . . what mode of operations are you using?”
B: ???

Cryptographic ciphers must be used in a proper mode of operation to ensure
- data privacy (confidentiality)
- data integrity (authentication)
Generic Composition

Authenticated Encryption Schemes

Modes of operations that ensure both privacy and integrity

Generic composition of secure encryption scheme and secure MAC usually leads to insecure AE schemes

[ Bellare Namprempre 2008 ]

“Building a secure crypto system is easy to do badly, and very difficult to do well”

– Bruce Schneier
There are a lot of beautiful AE schemes that are provably secure under *reasonable assumptions* (CWC, GCM, OCB,...) If you need encryption. Use them when possible. Please!
There are a lot of beautiful AE schemes that are provably secure under \textit{reasonable assumptions}\ (CWC, GCM, OCB,\ldots)

If you need encryption. Use them when possible. Please!
There are a lot of beautiful AE schemes that are provably secure under \textit{reasonable assumptions} (CWC, GCM, OCB,...)

If you need encryption. Use them when possible. Please!

**QA-Session**

Q: Are AE schemes misuse resistant, and will the honest developer apply it properly?

A: No! :-)

Christian Forler, Stefan Lucks, Jakob Wenzel
Designing the API for a Cryptographic Library
Proper encryption schemes are only secure under reasonable assumptions
Proper encryption schemes are only secure under reasonable assumptions.

Usually, cryptographers publish this assumptions only in cryptographic conferences and journals.
Nonce - Number Used Only Once

Modern AE schemes are not deterministic but nonce based

\[ E_K(P, N) = C_T \]

\[ E_K(P, N') = C'_T' \]
Nonce Misuse Issue

It is not unusual that $K$ and $N$ determine a keystream $S$. 

\[ F_K(N, P) = C \]

\[ F_K(N, P') = C' \]
Nonce Misuse Issue

It is not unusual that $K$ and $N$ determine a keystream $S$

\[ \begin{align*}
N &\quad P \\
\Rightarrow &\quad F_K \\
\quad &\quad C
\end{align*} \quad \begin{align*}
N &\quad P' \\
\Rightarrow &\quad F_K \\
\quad &\quad C'
\end{align*} \]

Fatal privacy issue, even for secure keystream generator $F_K$

\[ P \oplus P' = C \oplus C' \]

(Fatal integrity issues [ Fleischmann Forler Lucks 2012 ])
Nonce Reuse Examples

Examples of flawed implementations

- Intercepting Mobile Communications: The Insecurity of 802.11 [Borisov, Goldberg, Wagner 2001]
- The Misuse of RC4 in Microsoft Word and Excel [Wu 2005]
- Console Hacking 2010 - PS3 Epic Fail [Hotz 2010]
- ...

⇒ Even big players as Microsoft and Sony sometimes get it wrong
Nonce Reuse Prevention

Ada Countermeasure Against Nonce Reuse

A limited and private type that is always updated before reading
Proposed and Implemented Solution (ACL-0.5.4)

generic
  type Block is private;

package Crypto.Types.Nonce_Generator is

  type Nonce is abstract limited new ...;

  function Update(This : in out Nonce)
    return Block is abstract;

end Crypto.Types.Nonce_Generator;
Implementation of a Random Nonce Generator

```
function Update(This: in out Nonce_Rand) return ...
 Byte_Array: Bytes(0..(Block'Size / 8)−1);
begin
  Crypto.Types.Random.Read(Byte_Array);
  return To_Block_Type(Byte_Array)
end Update;
```
Implementation of a Random Nonce Generator

```plaintext
function Update(This: in out Nonce_Rand) return ...
    Byte_Array: Bytes(0 .. (Block'Size / 8) - 1);
begin
    Crypto.Types.Random.Read(Byte_Array);
    return To_Block_Type(Byte_Array)
end Update;
```

Collision probability for \( q \) invocation of the function \Update:\n
\[
\leq \frac{q^2}{2^n} \quad n = \text{Block'Size}
\]
## Supported Nonce Generators (ACL-0.5.4)

<table>
<thead>
<tr>
<th>Name</th>
<th>Random Source</th>
<th>NV Memory</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>No</td>
<td>Yes</td>
<td>Ctr</td>
</tr>
<tr>
<td>Random</td>
<td>Yes</td>
<td>No</td>
<td>R</td>
</tr>
<tr>
<td>Mixed-1</td>
<td>Yes</td>
<td>No</td>
<td>Ctr</td>
</tr>
<tr>
<td>Mixed-2</td>
<td>Yes</td>
<td>No</td>
<td>R and Ctr</td>
</tr>
</tbody>
</table>

Note that the implementation of a nonce type requires at least NV memory or a random source.
Plaintext Leaking Scenario

Definition (Plaintext Leaking)
Application stores (parts of) an unauthenticated plaintext
Plaintext Leaking Example

Decryption APIs usually process plain/ciphertext chunks

```
procedure Decrypt (C : in Ciphertext_Chunk;
                 P : out Plaintext_Chunk);
```

```
procedure Final_Decrypt (C : in Ciphertext_Chunk;
                        T : in Tag_T;
                        P : out Plaintext_Chunk;
                        Verified : out Boolean);
```

What happens if the Verification failed?
Plaintext Leaking Example

Decryption APIs usually process plain/ciphertext chunks

```
procedure Decrypt(C : in Ciphertext_Chunk; P : out Plaintext_Chunk);

procedure Final_Decrypt(C : in Ciphertext_Chunk; T : in Tag_T; P : out Plaintext_Chunk; Verified : out Boolean);
```

What happens if the Verification failed?

At least \( n - 1 \) chunks of the invalid Plaintext have been delivered to the application.
Application User Awareness

One out of five application user ignore security warnings [Egelman 2008 ]

Invalid Ciphertext

The validation of the ciphertext failed. The ciphertext might be modified by an evil adversary.

Proceed anyway Delete Plaintext
Goal

Plaintext Leaking Countermeasure

Design a proper API that never leak parts of unauthenticated plaintext to an application

Christian Forler, Stefan Lucks, Jakob Wenzel
Designing the API for a Cryptographic Library
**Goal**

**Plaintext Leaking Countermeasure**
Design a proper API that never leak parts of unauthenticated plaintext to an application.

**Drawback**
Usually, a ciphertext must be processed twice:
1. Authenticate ciphertext
2. Decrypt ciphertext
Our Solution (ACL-0.5.4)

```pascal
type AE_Scheme is limited interface;

type Writer is access procedure (B : in Bytes);

type Reader is access procedure
  (B : out Bytes; Count : out Natural);

function D_And_V (This : in out AE_Scheme;
  Ciphertext_F : in Reader;
  Ciphertext_S : in Reader := null;
  Plaintext : in Writer)
  return Boolean is abstract;
```
Summary

- We need more communication between academic theory and industrial practice
  - Cryptographers shall share their results with engineers
  - Engineers shall consult cryptographers when implementing crypto systems

A good cryptographic library should be useful for non-cryptographers and resistant to common misuse issues. What do you think about the Ada-Crypto-Library? (http://github.com/cforler/Ada-Crypto-Library) We are eager to hear from you.
We need more communication between academic theory and industrial practice

- **Cryptographers** shall share their results with engineers
- Engineers shall consult **cryptographers** when implementing crypto systems

A good cryptographic library should be

- useful for non cryptographers
- resistant to common misuse issues
Summary

- We need more communication between academic theory and industrial practice
  - Cryptographers shall share their results with engineers
  - Engineers shall consult cryptographers when implementing crypto systems
- A good cryptographic library should be
  - useful for non cryptographers
  - resistant to common misuse issues
- We are eager to hear from you
Questions?