What the Industry Really Really Needs

Roberto Avanzi

Qualcomm Product Security Germany
So Cryptography

Very Security

Much Lightweight
Disclaimer

This is not a technical talk.

(Except for the technical parts.)

(Which are often not cryptographic at all.)

(And if they are cryptographic they mostly deal with PKC.)
Disclaimer

This is not a technical talk.

(Except for the technical parts.)

(Which are often not cryptographic at all.)

(And if they are cryptographic they mostly deal with PKC.)
Sequel

This is also a sequel to last year’s talk.

But like any good sequel it is also a self contained story.
What the Industry Really Really Wants Needs
When does the industry care?

Companies do things that are good for their business.

Ex: Users trust and use a platform $\Rightarrow$ perception and reputation gain.

- Remember: security often an afterthought in mobile space.

- Lesson from 4G ciphers in China.
  4G ciphers implemented only at last moment.
  Only integrity, not confidentiality, initially supported.

  (Implementation: A. and Brumley, ISC 2013, ePrint 2013/428)

(Note: I will mark problems, as we stumble upon them, with a ⋄.)
When does the industry care?

Companies do things that are good for their business.

Ex: Users trust and use a platform \(\Rightarrow\) perception and reputation gain.

▶ Remember: security often an afterthought in mobile space.

▶ Lesson from 4G ciphers in China.

4G ciphers implemented only at last moment.

Only integrity, not confidentiality, initially supported.

(Implementation: A. and Brumley, ISC 2013, ePrint 2013/428)

(Note: I will mark problems, as we stumble upon them, with a ⭐.)
Why should I care about the needs of the industry?

I care about users.
Their happiness brings business and advantages to me.
Also I am one of them.

There are two types of users I am considering:

- End users.
- Content producers as customers/users of a platform/ecosystem (Content = SW, IP, media).

Focusing about the user’s needs is beneficial to the industry and to me.
Why should I care about the needs of the industry?

I care about users.

Their happiness brings business and advantages to me. Also I am one of them.

There are two types of users I am considering:

- End users.

- Content producers as customers/users of a platform/ecosystem (Content = SW, IP, media).

Focusing about the user’s needs is beneficial to the industry and to me.
Which problems am I concerned with?

Let us zoom out.

- Crypto is an indispensable component in a sound security architecture.
- And someone will have to implement this architecture, including the crypto.
  - Implementation can be both a source of trouble ...
  - ... and a means to goal (performance criteria).
- Also, a security architecture is part of a solution or product.

Zoom out again.
Which problems am I concerned with?

Let us zoom out.

- Crypto is an indispensable component in a sound security architecture.
- And someone will have to implement this architecture, including the crypto.
  - Implementation can be both a source of trouble ...
  - ... and a means to goal (performance criteria).
- Also, a security architecture is part of a solution or product.

Zoom out again.
Which problems am I concerned with?

Let us zoom out.

- Crypto is an indispensable component in a sound security architecture.
- And someone will have to implement this architecture, including the crypto.
  - Implementation can be both a source of trouble ...
  - ... and a means to goal (performance criteria).
- Also, a security architecture is part of a solution or product.

Zoom out again.
The whole solution must be light

And this is my guiding principle today.

Light on all resources, including human, including users.

- Building blocks in real world usage.
- How these blocks are used.
- How all is easy to implement correctly/securely.
- The solution’s usability.
So we have a list of needs already

Sorting and merging the starred concerns, in a personal fashion, and *going from top to bottom*, we have:

1. Lightweight security *solutions*.
2. “Real world efficient” implementation of cryptographic modules.
3. Ciphers *for* high-throughput systems (think 1000x networks).
4. Ciphers that *resist* implementation errors or sabotage.
5. Ciphers that *help* against exploitation.
Let's start!
Outline

Introduction

Lightweight Solutions

Implementing for the Real World

Efficient Ciphers

Tough Ciphers

Helpful Ciphers

Conclusion
One can approach this from the point of view of computational efficiency:

1. Need protocols for LW authentication and encryption for IoT, but also internet.

2. The progression from HTTP to HTTP/2, SPDY (RIP), TLS false start, are good steps in right reduction. Minimise overheads. (Orr mentioned Big Data. Let’s talk about small data. Why do we have to send 124 bytes worth of data for a single one byte message?)

3. For a different approach, see Bogdanov, Lauridsen, and Tischhauser, FSE 2015.

But there is a more important aspect, because most features have a user-facing side.
1. Lightweight Security Solutions 2/3

Efficient ID/security/privacy management MIA.

We need to make this easy. Light. For the user.

How many 15 year olds understand security?
How many 80 year olds understand security?

They do not know how to get security/privacy.
But, do we?

Look at the big picture: devise how solution should work, then proceed toward the lower levels.
This will drive the requirements for the cryptographic protocols, then primitives.
Efficient ID/security/privacy management MIA.

We need to make this easy. Light. For the user.

How many 15 year olds understand security?
How many 80 year olds understand security?

They do not know how to get security/privacy. But, do we?

Look at the big picture: devise how solution should work, then proceed toward the lower levels. This will drive the requirements for the cryptographic protocols, then primitives.
1. Lightweight Security Solutions 2/3

Efficient ID/security/privacy management MIA.

We need to make this easy. Light. For the user.

How many 15 year olds understand security?
How many 80 year olds understand security?

They do not know how to get security/privacy. But, do we?

Look at the big picture: devise how solution should work, then proceed toward the lower levels. This will drive the requirements for the cryptographic protocols, then primitives.
1. Lightweight Security Solutions 2/3

Efficient ID/security/privacy management MIA.

We need to make this easy. Light. For the user.

How many 15 year olds understand security?
How many 80 year olds understand security?

They do not know how to get security/privacy.
But, do we?

Look at the big picture: devise how solution should work, then proceed toward the lower levels.
This will drive the requirements for the cryptographic protocols, then primitives.
1. Lightweight Security Solutions 3/3

More of the same:

PGP/GPG is a usability disaster. Who uses it? Have you seen MacGPG?

If you want to create widespread awareness, a movement to create a market drive, the product must be easy to use.

Difficult: think like the user. User != RMS.

Design system from top to bottom.

Easier than designing a good GUI wrapper around GPG.

Not your field? Yes it is.

Precedent: cryptocat.
1. Lightweight Security Solutions 3/3

More of the same:

PGP/GPG is a usability disaster.  
Who uses it? Have you seen MacGPG?

If you want to create widespread awareness, a movement to create a market drive, the product must be easy to use.

Difficult: think like the user. User != RMS.

Design system from top to bottom.

Easier than designing a good GUI wrapper around GPG.

Not your field?  
Yes it is.

Precedent: cryptocat.
1. Lightweight Security Solutions 3/3

More of the same:

PGP/GPG is a usability disaster.
Who uses it? Have you seen MacGPG?

If you want to create widespread awareness, a movement to create a market drive, the product must be easy to use.

Difficult: think like the user. User != RMS.

Design system from top to bottom.

Easier than designing a good GUI wrapper around GPG.

Not your field?
Yes it is.

Precedent: cryptocat.
1. Lightweight Security Solutions 3/3

More of the same:

PGP/GPG is a usability disaster.
Who uses it? Have you seen MacGPG?

If you want to create widespread awareness, a movement to create a market drive, the product must be easy to use.

Difficult: think like the user. User != RMS.

Design system from top to bottom.

Easier than designing a good GUI wrapper around GPG.

Not your field?
Yes it is.

Precedent: cryptocat.
Outline

Introduction

Lightweight Solutions

Implementing for the Real World

Efficient Ciphers

Tough Ciphers

Helpful Ciphers

Conclusion
2. Implementing for the Real World 1/13

- A lot of people in industry do not understand performance. This leads to bloatware.

- A lot of people in academia do not understand performance.* This leads to speed record breaking software. Which is also bloatware.† And people in industry use it without understanding it.

* To be more precise, what level of performance industrial products need as related to other types of measurement, such as memory footprint.

† In a very specific context, of course. I will be talking about fantastic software, written by some of the best people in academia and industry – but that it is too easy to use in the wrong way.
2. Implementing for the Real World 1/13

► A lot of people in industry do not understand performance. This leads to bloatware.

► A lot of people in academia do not understand performance.* This leads to speed record breaking software. Which is also bloatware.† And people in industry use it without understanding it.

* To be more precise, what level of performance industrial products need as related to other types of measurement, such as memory footprint.

† In a very specific context, of course. I will be talking about fantastic software, written by some of the best people in academia and industry – but that it is too easy to use in the wrong way.
2. Implementing for the Real World 2/13

Modified from: http://flickr.com/photos/11939863@N08/3793288383, by SoraZG. Licensed under the terms of the cc-by-2.0.
2. Implementing for the Real World 3/13

Yes, NaCl.

Written by djb (djb!) in qhasm. Speed records! Secure!

There is a portable C version, called libsodium.

Apple uses curve25519, ed25519, ChaCha20, and Poly1305 in HomeKit.

Everybody in IoT needs to implement these ciphers.

libodium is PD, curve25519 and ed25519 based on ref10 (djb!) and AGL’s code (AGL!).

Code written by djb!!! AGL!!!

So everybody is using it verbatim for IoT devices.

And since this is code from great guys, this must be the best situation evah!
Yes, NaCl.

Written by djb (djb!) in qhasm. Speed records! Secure!

There is a portable C version, called libsodium.

Apple uses curve25519, ed25519, ChaCha20, and Poly1305 in HomeKit.

Everybody in IoT needs to implement these ciphers.

libsodium is PD, curve25519 and ed25519 based on ref10 (djb!) and AGL’s code (AGL!).

Code written by djb!!! AGL!!!

So everybody is using it verbatim for IoT devices.

And since this is code from great guys, this must be the best situation evah!
2. Implementing for the Real World 3/13

Yes, NaCl.

Written by djb (djb!) in qhasm. Speed records! Secure!

There is a portable C version, called libsodium.

Apple uses curve25519, ed25519, ChaCha20, and Poly1305 in HomeKit.

Everybody in IoT needs to implement these ciphers.

libsodium is PD, curve25519 and ed25519 based on ref10 (djb!) and AGL’s code (AGL!).

Code written by djb!!! AGL!!!

So everybody is using it verbatim for IoT devices.

And since this is code from great guys, this must be the best situation evah!
2. Implementing for the Real World 3/13

Yes, NaCl.

Written by djb (djb!) in qhasm. Speed records! Secure!

There is a portable C version, called libsodium.

Apple uses curve25519, ed25519, ChaCha20, and Poly1305 in HomeKit.

Everybody in IoT needs to implement these ciphers.

libsodium is PD, curve25519 and ed25519 based on ref10 (djb!) and AGL’s code (AGL!).

Code written by djb!!! AGL!!!

So everybody is using it verbatim for IoT devices.

And since this is code from great guys, this must be the best situation evah!
2. Implementing for the Real World
2. Implementing for the Real World 5/13

NOPE
2. Implementing for the Real World 6/13
Let us have a look at the signature:

- Single scalar multiplication code 42 KB, of which 30 KB static data.
- Signature scalar muladd/reduction code: unrolled 47 KB bytes.

The scalar multiplication code is a comb method. To compute \( z \cdot P \), first write scalar as

\[
z = r_0 + 16 \cdot r_1 + 16^2 \cdot r_2 + \cdots + 16^{63} \cdot r_{63},
\]

with \( 1 \leq |r_i| \leq 8 \). With precomputations \( 16^i \cdot r \cdot P \) for \( 0 \leq i \leq 63 \) and \( 1 \leq r \leq 8 \), calculate

\[
z \cdot P = \sum_{i=0}^{63} r_i \cdot 16^i \cdot P = 2^4 \cdot \left( \sum_{1 \leq i \leq 63, \text{odd}} r_i \cdot 16^{i-1} \cdot P \right) + \left( \sum_{0 \leq i \leq 62, \text{even}} r_i \cdot 16^i \cdot P \right).
\]

Tables precomputed only for even \( i \) \( \Rightarrow \) Horner scheme with 4 point doublings.
So 32 tables (not 64), 8 entries each, one point is 3 coordinates, 10 bytes each, 30720 bytes.
2. Implementing for the Real World 7/13

Let us have a look at the signature:

- Single scalar multiplication code 42 KB, of which 30 KB static data.
- Signature scalar muladd/reduction code: unrolled 47 KB bytes.

The scalar multiplication code is a comb method. To compute $z \cdot P$, first write scalar as

$$z = r_0 + 16 r_1 + 16^2 r_2 + \cdots + 16^{63} r_{63},$$

with $1 \leq |r_i| \leq 8$. With precomputations $16^i r \cdot P$ for $0 \leq i \leq 63$ and $1 \leq r \leq 8$, calculate

$$z \cdot P = \sum_{i=0}^{63} r_i 16^i \cdot P = 2^4 \cdot \left( \sum_{1 \leq i \leq 63} r_i 16^{i-1} \cdot P \right) + \left( \sum_{0 \leq i \leq 62} r_i 16^i \cdot P \right).$$

Tables precomputed only for even $i \Rightarrow$ Horner scheme with 4 point doublings.

So 32 tables (not 64), 8 entries each, one point is 3 coordinates, 10 bytes each, 30720 bytes.
2. Implementing for the Real World 8/13

Too much for a lightbulb with a tiny Tensilica CPU, and all the other IoT SW stacks.

If you have to fit all stacks in, say, 1 MB, you cannot waste 300 KB for a few primitives.

We can compute instead the tables for $8 | i$, use 12 doublings.
Slowdown 12-20% on target for signature generation (short message).
Static data: only 7680 bytes.
(If only $16 | i$, 28 doublings, slowdown 20-30%, 3840 bytes static data only.)

This is not a trivial optimisation if you are not a cryptographer.
I looked at the binaries from four sources, they all use 30720 bytes.

You can save another 960 bytes, slowing down signature verification $\sim 1\%$. 
2. Implementing for the Real World 9/13

*But there is more!*

In $\text{ed25519}$ you need to compute $a \cdot b + c \mod \text{group order}$ (not $p = 2^{255} - 19$).

The cost of such an operation is about two field mults.

One scalar multiplication of the fixed base point costs about $800 - 840$ field muls.

Field multiplication is fully unrolled. Scalar manipulation code fully unrolled as well.

This in AGL’s code and *all subsequent branches* because they just translated djb to C.*

De-unroll it: 11.444 bytes. 36K of code saved.

This particular operation becomes 4-5 times slower, but:

*Impact on the crypto operations: just ~ 2% for signing.*

*They did a terrific job, preserving most of the performance. But once you leave the original code’s territory, must absolute performance be your first priority?*
2. Implementing for the Real World 10/13

But there is even more!

libsodium keeps name spaces separated for each primitive.

curve25519 and ed25519 use the same field arithmetic.

It is exactly the same code, with the same API: A redundancy of about 95 KB.

With some more tweaks, you can end up saving about 200 KB of code+data, Slowdown of 12-20% in signature generation, 2% in verification, no impact on DH.

The implementation can be brought from about 300 KB to around 100 KB. Still big, but much better.

In a few hours for a cryptographer (that knows how to code).
2. Implementing for the Real World 10/13

*But there is even more!*

libsodium keeps name spaces separated for each primitive.

curve25519 and ed25519 use the same field arithmetic.

It is exactly the same code, with the same API: A redundancy of about 95 KB.

With some more tweaks, you can end up saving about 200 KB of code+data, Slowdown of 12-20% in signature generation, 2% in verification, no impact on DH.

The implementation can be brought from about 300 KB to around 100 KB. Still big, but much better.

In a few hours for a cryptographer (that knows how to code).
2. Implementing for the Real World 12/13

**Remark.** In Haifa there was a very interesting question from the audience: “Could the compiler, with some link time optimisation, recognise identical functions and merge them?”

Well, in fact we can do **better**!

At the 2013 LLVM Developers’ Meeting (http://llvm.org/devmtg/2013-11/) Tobias Edler von Koch and Pranav Bhandarkar of QuIC (Qualcomm Innovation Center) showed how a compiler can even merge *similar* functions.

But such features need to be used very carefully:

Imagine you have written code with no data-dependent branches.  
*Then the compiler takes similar functions, rewrites and merged them, and you end up with the data-dependent branches you wanted to avoid in the first place?*

Programming secure code is still about the programmer’s skills, a field where “let the compiler help us” can be more dangerous than helpful.

(Auto-merging the libsodium duplicate functions would be ok, though.)
2. Implementing for the Real World 12/13

Moral of the story:

- If you are a great implementor: do not assume you understand everything.
- If you are a great cryptographer: do not assume you understand everything.
- In general: do not assume you understand everything.
  (I have made myself this mistake in the past. It’s not fun.)
- If you are writing reference code, a public domain library for everybody, think at real world applications. Most changes I have described are good for high-end CPUs as well in real world applications (less I-cache and D-cache pressure).
- “Programming is thinking, not typing.” – Casey Patton
  (“You are scientists, not secretaries.” – me)
Moral of the story:

- If you are a great implementor: do not assume you understand everything.
- If you are a great cryptographer: do not assume you understand everything.
- In general: do not assume you understand everything.
  (I have made myself this mistake in the past. It’s not fun.)
- If you are writing reference code, a public domain library for everybody, think at real world applications. Most changes I have described are good for high-end CPUs as well in real world applications (less I-cache and D-cache pressure).
- “Programming is thinking, not typing.” – Casey Patton
  (“You are scientists, not secretaries.” – me)
2. Implementing for the Real World 12/13

Moral of the story:

- If you are a great implementor: do not assume you understand everything.

- If you are a great cryptographer: do not assume you understand everything.

- In general: do not assume you understand everything.
  (I have made myself this mistake in the past. It’s not fun.)

- If you are writing reference code, a public domain library for everybody, think at real world applications. Most changes I have described are good for high-end CPUs as well in real world applications (less I-cache and D-cache pressure).

- “Programming is thinking, not typing.” – Casey Patton
  (“You are scientists, not secretaries.” – me)
2. Implementing for the Real World 12/13

Moral of the story:

▶ If you are a great implementor: do not assume you understand everything.

▶ If you are a great cryptographer: do not assume you understand everything.

▶ In general: do not assume you understand everything.
  (I have made myself this mistake in the past. It’s not fun.)

▶ If you are writing reference code, a public domain library for everybody, think at real world applications. Most changes I have described are good for high-end CPUs as well in real world applications (less I-cache and D-cache pressure).

▶ “Programming is thinking, not typing.” – Casey Patton
  (“You are scientists, not secretaries.” – me)
2. Implementing for the Real World 12/13

Moral of the story:

- If you are a great implementor: do not assume you understand everything.
- If you are a great cryptographer: do not assume you understand everything.
- In general: do not assume you understand everything.
  (I have made myself this mistake in the past. It’s not fun.)
- If you are writing reference code, a public domain library for everybody, think at real world applications. Most changes I have described are good for high-end CPUs as well in real world applications (less I-cache and D-cache pressure).
- “Programming is thinking, not typing.” – Casey Patton
  (“You are scientists, not secretaries.” – me)
2. Implementing for the Real World 13/13

Very important:

Of course we must mention Peter Schwabe’s μNaCl on various micro-controllers and TweetNaCl.

They are great pieces of software, with focus on the small total footprint.

What I am concerned here is the *middle ground* and the fact that a lot of users use something that may not be ideal for them blindingly.
Outline

Introduction

Lightweight Solutions

Implementing for the Real World

Efficient Ciphers

Tough Ciphers

Helpful Ciphers

Conclusion
3. Ciphers for efficient implementation

Go beyond cipher + mode of operation. Already a lot of research in this direction (AEGIS, ALE... CAESAR competition).

Consider a cipher you can break with, say, $2^{32}$ KP/CP for same key, and you have a proof that you cannot break it with less data.

Refresh key every $2^{16}$ (use hash barrier, or a weaker-than-hash-but-ok-for-this-purpose).

Take/tweak/design such a cipher. Ditto for hash barrier.

Let the community analyse the security.

Doing a PhD? Start analysing what happens with reduced round AES and PRINCE.

(Disclaimer: such a cipher may easily not satisfy the requirements of next section.)
3. Ciphers for efficient implementation

Go beyond cipher + mode of operation.
Already a lot of research in this direction (AEGIS, ALE... CAESAR competition).

Consider a cipher you can break with, say, $2^{32}$ KP/CP for same key, and you have a proof that you cannot break it with less data.

Refresh key every $2^{16}$ (use hash barrier, or a weaker-than-hash-but-ok-for-this-purpose).

Take/tweak/design such a cipher. Ditto for hash barrier.

Let the community analyse the security.

Doing a PhD? Start analysing what happens with reduced round AES and PRINCE.

(Disclaimer: such a cipher may easily not satisfy the requirements of next section.)
Outline

Introduction

Lightweight Solutions

Implementing for the Real World

Efficient Ciphers

Tough Ciphers

Helpful Ciphers

Conclusion
4. Ciphers that resist implementation errors or sabotage 1/3

djb has recently talked about *Error-Prone Cryptographic Designs* (RWC 2015).

- Why should a protocol/signature scheme/mode of operation break *completely* if randomness is not good, instead of just giving up some property?
- Take: design protocols to not rely on it if not strictly necessary.
- Also, design your ciphers in a way that it is less difficult to break them exploiting implementation problems (cache attacks).
- djb’s advice to crypto designers:
  Creating or evaluating a design? Think about the implementations.
- Remember: As @ioerror just reminded us at FSE 2015, the NSA *does* think about the implementation when they push for this or that algorithm!
- Make everything constant-time and fast (and make parts non-constant time and faster only if you are 100% sure you can do that in a specific context).
4. Ciphers that resist implementation errors or sabotage 2/3

Bruce Schneier, Matthew Fredrikson, Tadayoshi Kohno, and Thomas Ristenpart:

**Surreptitiously Weakening Cryptographic Systems**

IACR eprint 2015/097

**Abstract:** Revelations over the past couple of years highlight the importance of understanding malicious and surreptitious weakening of cryptographic systems. *We provide an overview of this domain, using a number of historical examples to drive development of a weaknesses taxonomy. This allows comparing different approaches to sabotage. We categorize a broader set of potential avenues for weakening systems using this taxonomy, and discuss what future research is needed to provide sabotage-resilient cryptography.*

They mention *implementation fragility, nonce/IV reuse and bad randomness.*

New one: forget to fully describe, analyse and mandate *state machines* for your protocols.
Little example: in order to prevent cache attacks on SW implementations of block ciphers we cannot have data-dependent read locations. We know this from AES.

Solution: Read the whole table? Too expensive.

Another idea: Read the whole table.

For a small cipher, with a small sbox value $\rightarrow$ output,

```
#define SBOX 0x4D5E087619CA23FB

output = (SBOX >> (4*value)) & 0xf;
```

So, why not use it all to create a $4 \times 64$ bit sbox?

```
#define ROR64(x,b) (((x) >> (b)) | ((x) << (64-(b))) )

output = ROR64(SBOX,4*value);
```

I am working on this. Want to join forces?
4. Ciphers that resist implementation errors or sabotage 3/3

Little example: in order to prevent cache attacks on SW implementations of block ciphers we cannot have data-dependent read locations. We know this from AES.

Solution: Read the whole table? Too expensive.

Another idea: Read the whole table.

For a small cipher, with a small sbox value $\mapsto$ output,

```c
#define SBOX 0x4D5E087619CA23FB
output = (SBOX >> (4*value)) & 0xf;
```

So, why not use it all to create a $4 \times 64$ bit sbox?

```c
#define ROR64(x,b) ((x) >> (b)) | ((x) << (64-(b)))
output = ROR64(SBOX,4*value);
```

I am working on this. Want to join forces?
4. Ciphers that resist implementation errors or sabotage 3/3

Little example: in order to prevent cache attacks on SW implementations of block ciphers we cannot have data-dependent read locations. We know this from AES.

Solution: Read the whole table? *Too expensive.*

Another idea: Read the whole table.
For a small cipher, with a small sbox value $\mapsto$ output,

```c
#define SBOX 0x4D5E087619CA23FB

output = (SBOX >> (4*value)) & 0xf;
```

So, why not use it all to create a $4 \times 64$ bit sbox?

```c
#define ROR64(x,b) ( ((x) >> (b)) | ((x) << (64-(b))) )

output = ROR64(SBOX,4*value);
```

I am working on this. Want to join forces?
4. Ciphers that resist implementation errors or sabotage 3/3

Little example: in order to prevent cache attacks on SW implementations of block ciphers we cannot have data-dependent read locations. We know this from AES.

Solution: Read the whole table? Too expensive.

Another idea: Read the whole table.
For a small cipher, with a small sbox value $\mapsto$ output,

```c
#define SBOX 0x4D5E087619CA23FB
output = (SBOX >> (4*value)) & 0xf;
```

So, why not use it all to create a $4 \times 64$ bit sbox?

```c
#define ROR64(x,b) (((x) >> (b)) | ((x) << (64-(b))))
output = ROR64(SBOX,4*value);
```

I am working on this. Want to join forces?
Outline

Introduction

Lightweight Solutions

Implementing for the Real World

Efficient Ciphers

Tough Ciphers

Helpful Ciphers

Conclusion
5. Ciphers that help against software exploitation 1/7

Deja vu from LWD 2014.

We often hear: SW exploits bring down crypto implementations.

But why not use crypto to harden software security?

Example: hackers run some SW that in, say, 2 seconds, exploits a vulnerability in your device to deliver and run an arbitrary payload (unlock, JB...).

Would it be enough to make their efforts $2^{10}$, $2^{20}$, $2^{30}$ times slower?
Would it deter them?
Would it deter their customers/users?

Attach your phone to your PC. Click the `\`unlock\'` button and in about 2 secs the device will reboot.
In the meantime, do not remove the cable, do not switch off your PC or your phone.
5. Ciphers that help against software exploitation 1/7

Deja vu from LWD 2014.

We often hear: SW exploits bring down crypto implementations.

But why not use crypto to harden software security?

Example: hackers run some SW that in, say, 2 seconds, exploits a vulnerability in your device to deliver and run an arbitrary payload (unlock, JB...).

Would it be enough to make their efforts $2^{10}$, $2^{20}$, $2^{30}$ times slower?
Would it deter them?
Would it deter their customers/users?

Attach your phone to your PC. Click the `\'unlock\'` button and in about 2 secs the device will reboot.
In the meantime, do not remove the cable, do not switch off your PC or your phone.
5. Ciphers that help against software exploitation 1/7

Deja vu from LWD 2014.

We often hear: SW exploits bring down crypto implementations.

But why not use crypto to harden software security?

Example: hackers run some SW that in, say, 2 seconds, exploits a vulnerability in your device to deliver and run an arbitrary payload (unlock, JB...).

Would it be enough to make their efforts $2^{10}$, $2^{20}$, $2^{30}$ times slower?
Would it deter them?
Would it deter their customers/users?

Attach your phone to your PC. Click the `\`unlock`' button and in about 2 secs the device will reboot.

In the meantime, do not remove the cable, do not switch off your PC or your phone.
5. Ciphers that help against software exploitation 1/7

*Deja vu from LWD 2014.*

We often hear: SW exploits bring down crypto implementations.

But why not use crypto to harden software security?

Example: hackers run some SW that in, say, 2 seconds, exploits a vulnerability in your device to deliver and run an arbitrary payload (unlock, JB...).

Would it be enough to make their efforts $2^{10}$, $2^{20}$, $2^{30}$ times slower?

Would it deter them?

Would it deter their customers/users?

*Attach your phone to your PC. Click the `\`unlock\'` button and in about 2 secs the device will reboot.*

*In the meantime, do not remove the cable, do not switch off your PC or your phone.*
5. Ciphers that help against software exploitation 1/7

Deja vu from LWD 2014.

We often hear: SW exploits bring down crypto implementations.

But why not use crypto to harden software security?

Example: hackers run some SW that in, say, 2 seconds, exploits a vulnerability in your device to deliver and run an arbitrary payload (unlock, JB...).

Would it be enough to make their efforts $2^{10}$, $2^{20}$, $2^{30}$ times slower?
Would it deter them?
Would it deter their customers/users?

Attach your phone to your PC. Click the `\`unlock\'` button and in about 2 secs the device will reboot.
In the meantime, do not remove the cable, do not switch off your PC or your phone.
Deja vu from LWD 2014.

We often hear: SW exploits bring down crypto implementations.

But why not use crypto to harden software security?

Example: hackers run some SW that in, say, 2 seconds, exploits a vulnerability in your device to deliver and run an arbitrary payload (unlock, JB...).

Would it be enough to make their efforts $2^{10}$, $2^{20}$, $2^{30}$ times slower? Would it deter them? Would it deter their customers/users?

Attach your phone to your PC. Click the `\`unlock\' button and in about 2,000 secs the device will reboot. In the meantime, do not remove the cable, do not switch off your PC or your phone.
5. Ciphers that help against software exploitation 1/7

Deja vu from LWD 2014.

We often hear: SW exploits bring down crypto implementations.

But why not use crypto to harden software security?

Example: hackers run some SW that in, say, 2 seconds, exploits a vulnerability in your device to deliver and run an arbitrary payload (unlock, JB...).

Would it be enough to make their efforts $2^{10}$, $2^{20}$, $2^{30}$ times slower? Would it deter them? Would it deter their customers/users?

Attach your phone to your PC. Click the `\texttt{\textasciitilde unlock}' button and in about 2,000,000 secs the device will reboot.

In the meantime, do not remove the cable, do not switch off your PC or your phone.
Deja vu from LWD 2014.

We often hear: SW exploits bring down crypto implementations.

But why not use crypto to harden software security?

Example: hackers run some SW that in, say, 2 seconds, exploits a vulnerability in your device to deliver and run an arbitrary payload (unlock, JB...).

Would it be enough to make their efforts $2^{10}$, $2^{20}$, $2^{30}$ times slower?
Would it deter them?
Would it deter their customers/users?

*Attach your phone to your PC. Click the `\`unlock\' button and in about 2,000,000,000 secs the device will reboot.*

*In the meantime, do not remove the cable, do not switch off your PC or your phone.*
5. Ciphers that help against software exploitation 2/7

Mike Frantzen and Mike Shuey, Usenix Security ’01: StackGhost.
XOR the stack frame but mention encryption as well:

“It could be a cryptographically weak usage but would stop all
but the most determined adversaries”

Return address encryption in software.

Stack frame encryption using HW support and binary modification.

“light-weight encryption hardware techniques [...] can be used to
provide protection with little performance overhead.”

(One of their “lightweight” ciphers: 4 round Feistel cipher
with the full AES as the round function. Yeah, sure.)
5. Ciphers that help against software exploitation 2/7

Mike Frantzen and Mike Shuey, Usenix Security ’01: StackGhost.
XOR the stack frame but mention encryption as well:

“It could be a cryptographically weak usage but would stop all but the most determined adversaries”

Return address encryption in software.

Stack frame encryption using HW support and binary modification.

“light-weight encryption hardware techniques [...] can be used to provide protection with little performance overhead.”

(One of their “lightweight” ciphers: 4 round Feistel cipher with the full AES as the round function. Yeah, sure.)
5. Ciphers that help against software exploitation 2/7

Mike Frantzen and Mike Shuey, Usenix Security ’01: StackGhost.
XOR the stack frame but mention encryption as well:

“It could be a cryptographically weak usage but would stop all but the most determined adversaries”

Return address encryption in software.

Stack frame encryption using HW support and binary modification.

“light-weight encryption hardware techniques [...] can be used to provide protection with little performance overhead.”

(One of their “lightweight” ciphers: 4 round Feistel cipher with the full AES as the round function. Yeah, sure.)
5. Ciphers that help against software exploitation 2/7

Mike Frantzen and Mike Shuey, Usenix Security ’01: StackGhost. XOR the stack frame but mention encryption as well:

“It could be a cryptographically weak usage but would stop all but the most determined adversaries”


“light-weight encryption hardware techniques [...] can be used to provide protection with little performance overhead.”

(One of their “lightweight” ciphers: 4 round Feistel cipher with the full AES as the round function. Yeah, sure.)
5. Ciphers that help against software exploitation 3/7

Now comes the FUN part.

Observations: you do not need to encrypt addresses.

While some code is executing, until it returns or calls back, it is in control.

So you can just *verify* these addresses in the code itself (google CFI, LLVM CFI).

Now, let us use crypto.

We know that if you can generate 1 bit securely, you can achieve any level of security.

So why not, say, 7 bits?

Let us use a 7 bit keyed and tweaked MAC for pointer addresses.
5. Ciphers that help against software exploitation 3/7

Now comes the FUN part.

Observations: you do not need to encrypt addresses.

While some code is executing, until it returns or calls back, it is in control.

So you can just verify these addresses in the code itself (google CFI, LLVM CFI).

Now, let us use crypto.

We know that if you can generate 1 bit securely, you can achieve any level of security.

So why not, say, 7 bits?

Let us use a 7 bit keyed and tweaked MAC for pointer addresses.
5. Ciphers that help against software exploitation 4/7

Roberto Avanzi: What the Industry Really Really Needs
Return oriented programming is about borrowing code chunks from an existing program or library and executing them instead of executing code on the stack.

1. Analyse the code chunks in the functions: Collect these gadgets.
2. Enough gadgets will most likely be (sufficiently close to) Turing complete. So, concatenating many of them you can run what you want.
3. Prepare information on the stack that will be processed by these operations.
4. Take into account the final stack pointer adjustment.
5. Also make sure the popped link register points to the next gadget.
6. Now, smash this sequence of stack frames on the stack.
7. RETURN. This starts a domino effect.
8. It is just threaded code.
5. Ciphers that help against software exploitation 6/7

In ROP you return from functions you did not enter properly

**Function prologue**

```c
void f(const char *inbuf, int len) {
    char buf[8];
    ▶ SP -= 0x40
    ▶ insert TAG(LR,key,SP) in LR
    ▶ store FP and LR on stack
    ▶ FP = SP + 0x30
}
```

**Function epilogue**

```c
    ▶ load FP and LR from stack
    ▶ verify TAG in LR using SP as “tweak”
    ▶ if TAG wrong then panic
    ▶ delete TAG bit field in LR
    ▶ SP += 0x40
    ▶ RETURN
```

The tag is inserted into unused bits of the address.
5. Ciphers that help against software exploitation 6/7

In ROP you return from functions you did not enter properly

**Function prologue**

```c
void f(const char *inbuf, int len) {
    char buf[8];
    SP -= 0x40
    insert TAG(LR, key, SP) in LR
    store FP and LR on stack
    FP = SP + 0x30
}
```

**Function epilogue**

```c
load FP and LR from stack
verify TAG in LR using SP as “tweak”
if TAG wrong then panic
delete TAG bit field in LR
SP += 0x40
RETURN
```

The tag is inserted into unused bits of the address.
5. Ciphers that help against software exploitation 7/7

ROP becomes possible if the attacked can guess the TAG. Suppose TAG is 7 bits.

Brute force: for 7 bits it is 128 choices.

A ROP payload consists of several gadgets.

If you have to guess 10 tags, then you must guess 70 bits.

If you assume you guessed the first 7 bit correctly, the space for key and tweak may be reduced somewhat. And you know something about the tweak (it is the SP).

For each additional “guessed” tag choices may be further reduced.

Determine correct adversarial model, design primitive.
## Conclusion

- Real world issues that affect the whole industry can lead to interesting problems in or around cryptography, new applications.

- I chose a few that I consider very important, related to my own work as a security engineer in Qualcomm’s product security team.

- Often not trivial.

- Different from “traditional” (lightweight) crypto questions.

- Exercise: Find more going from top to bottom.
Industry leading security team:

“You know who has a shockingly competent security team? Qualcomm.”

Dan Kaminsky

https://twitter.com/dakami/statuses/229057109566832641
הדות