Crypto that is Light to Accept

Roberto Avanzi

Qualcomm Product Security Germany

Haifa, Technion, March 28, 2016 (Slides Revised April 20, 2016)

Outline

Wow

So Cryptography

Very Security

Much Lightweight

Disclaimer

This *is* a technical talk.

And I give some answers to questions asked in 2014 and 2015.

Disclaimer

This *is* a technical talk.

And I give some answers to questions asked in 2014 and 2015.

How does the real world handle security/crypto?

- Remember: security used to be an afterthought in many spaces.
- In the past: Design everything, do not even consider crypto and security aspects.
- But it is getting increasingly important every day.
- Hence: Design everything, including a provision for crypto and security aspects.
  When 99% of resources and time have been used, ask the company cryptographer and *assume* she *will* use some off-the-shelf solution that fits the bill.
- Assumption: Cryptographers, like ma [themati | gi] cians, work with arcane symbols. Hence they can do impossible stuff.

  Sometimes they even present new methods that come together with a security proof. But since the new method is new, it will (usually) be rejected.

  (Unless, in a few cases, when there is no alternative.)

How does the real world handle security/crypto?

- Remember: security used to be an afterthought in many spaces.
- In the past: Design everything, do not even consider crypto and security aspects.
- But it is getting increasingly important every day.
- Hence: Design everything, including a provision for crypto and security aspects. When 99% of resources and time have been used, ask the company cryptographer and *assume* she *will* use some off-the-shelf solution that fits the bill.
- Assumption: Cryptographers, like ma [themati | gi] cians, work with arcane symbols. Hence they can do impossible stuff.

  Sometimes they even present new methods that come together with a security proof. But since the new method is new, it will (usually) be rejected.

  (Unless, in a few cases, when there is no alternative.)

How does the real world handle security/crypto?

- Remember: security used to be an afterthought in many spaces.
- In the past: Design everything, do not even consider crypto and security aspects.
- But it is getting increasingly important every day.
- Hence: Design everything, including a provision for crypto and security aspects.
  When 99% of resources and time have been used, ask the company cryptographer and *assume* she *will* use some off-the-shelf solution that fits the bill.
- Assumption: Cryptographers, like ma [themati | gi] cians, work with arcane symbols. Hence they can do impossible stuff.

  Sometimes they even present new methods that come together with a security proof. But since the new method is new, it will (usually) be rejected.

  (Unless, in a few cases, when there is no alternative.)

How does the real world handle security/crypto?

- Remember: security used to be an afterthought in many spaces.
- In the past: Design everything, do not even consider crypto and security aspects.
- But it is getting increasingly important every day.
- Hence: Design everything, including a provision for crypto and security aspects. When 99% of resources and time have been used, ask the company cryptographer and *assume* she *will* use some off-the-shelf solution that fits the bill.
- Assumption: Cryptographers, like ma [themati | gi] cians, work with arcane symbols. Hence they can do impossible stuff.

  Sometimes they even present new methods that come together with a security proof. But since the new method is new, it will (usually) be rejected.

  (Unless, in a few cases, when there is no alternative.)

How does the real world handle security/crypto?

- Remember: security used to be an afterthought in many spaces.
- In the past: Design everything, do not even consider crypto and security aspects.
- But it is getting increasingly important every day.
- Hence: Design everything, including a provision for crypto and security aspects. When 99% of resources and time have been used, ask the company cryptographer and *assume* she *will* use some off-the-shelf solution that fits the bill.
- Assumption: Cryptographers, like ma [themati | gi] cians, work with arcane symbols. Hence they can do impossible stuff.

  Sometimes they even present new methods that come together with a security proof. But since the new method is new, it will (usually) be rejected.

  (Unless, in a few cases, when there is no alternative.)

How does the real world handle security/crypto?

- Remember: security used to be an afterthought in many spaces.
- In the past: Design everything, do not even consider crypto and security aspects.
- But it is getting increasingly important every day.
- Hence: Design everything, including a provision for crypto and security aspects. When 99% of resources and time have been used, ask the company cryptographer and *assume* she *will* use some off-the-shelf solution that fits the bill.
- Assumption: Cryptographers, like ma [themati | gi] cians, work with arcane symbols. Hence they can do impossible stuff.

  Sometimes they even present new methods that come together with a security proof. But since the new method is new, it will (usually) be rejected.

  (Unless, in a few cases, when there is no alternative.)

Generalisation

1. There's a need for crypto primitives that are lighter (on some metric) than, say, general purpose primitives like AES, SHA-2, NIST EC...
2. Crypto community develops *lightweight* solutions for *specific* applications.
3. Lightweight is *new*.
4. New is by definition (not always) unproven - now generalise: Always unproven.
5. Unproven is bad.
6. Hence lightweight is bad.

New is always better

(Barney Stinson)

New is always worse

(Customer looking at lightweight crypto)

I will tell you two stories

A short one and a longer one

They are examples of the difficulty to accept new cryptographic solutions and how you work around them trying to be creative at the same time

1. My personal curve
2. My QARMA (sic) will come back to haunt me

The first story has a bad ending
The second one is still work in progress but should have a happy end

Outline

Banking Industry

At a previous job, I consulted for a banking/CC consortium
The Customer wanted secure cards for banking

Cards needed to be able to *generate* digital signatures
We persuaded them to use ECC instead of RSA
Advantage: no need to spend considerable time on prime generation on an expensive provisioning server (cost on provisioning ground was a problem with RSA keygen)
One curve fits all! Lightweight! (Remember: lightweight is a relative concept.)

> *No, wait*, they said, *each RSA key has its own modulus!*
> *Therefore, each card needs its own, unique, secure curve!*

We tried to explain again that you use:

> one curve for *all* cards, a unique secret multiplier *per card* ...
> ... no need for expensive generation of *many* cryptographically secure elliptic curves ...

They went away, still not understanding and disappointed ...
... and the product never saw the light

Banking Industry

At a previous job, I consulted for a banking/CC consortium
The Customer wanted secure cards for banking

Cards needed to be able to *generate* digital signatures
We persuaded them to use ECC instead of RSA
Advantage: no need to spend considerable time on prime generation on an expensive provisioning server (cost on provisioning ground was a problem with RSA keygen)
One curve fits all! Lightweight! (Remember: lightweight is a relative concept.)

No, wait, they said, *each RSA key has its own modulus!*
Therefore, each card needs its own, unique, secure curve!

We tried to explain again that you use:
one curve for *all* cards, a unique secret multiplier *per card* ...
... no need for expensive generation of *many* cryptographically secure elliptic curves ...

They went away, still not understanding and disappointed ...
... and the product never saw the light

Banking Industry

At a previous job, I consulted for a banking/CC consortium
The Customer wanted secure cards for banking

Cards needed to be able to *generate* digital signatures
We persuaded them to use ECC instead of RSA
Advantage: no need to spend considerable time on prime generation on an expensive provisioning server (cost on provisioning ground was a problem with RSA keygen)
One curve fits all! Lightweight! (Remember: lightweight is a relative concept.)

> *No, wait*, they said, *each RSA key has its own modulus!*
> *Therefore, each card needs its own, unique, secure curve!*

We tried to explain again that you use:

> one curve for *all* cards, a unique secret multiplier *per card* ...
> ... no need for expensive generation of *many* cryptographically secure elliptic curves ...

They went away, still not understanding and disappointed ...
... and the product never saw the light

Banking Industry

At a previous job, I consulted for a banking/CC consortium
The Customer wanted secure cards for banking

Cards needed to be able to *generate* digital signatures
We persuaded them to use ECC instead of RSA
Advantage: no need to spend considerable time on prime generation on an expensive provisioning server (cost on provisioning ground was a problem with RSA keygen)
One curve fits all! Lightweight! (Remember: lightweight is a relative concept.)

> *No, wait*, they said, *each RSA key has its own modulus!*
> *Therefore, each card needs its own, unique, secure curve!*

We tried to explain again that you use:

> $\underline{\text{one}}$ curve for *all* cards, a $\underline{\text{unique}}$ secret multiplier *per card* ...
> ... $\underline{\text{no need}}$ for expensive generation of *many* cryptographically secure elliptic curves ...

They went away, still not understanding and disappointed ...
... and the product never saw the light

Orr asked “why now?”

But when these things happen...

You ask yourself: “*why me?*”

Outline

Two (plus one) use cases for a lightweight cipher

Memory Encryption

Pointer Authentication

Very Low Security Hashes

Memory Encryption

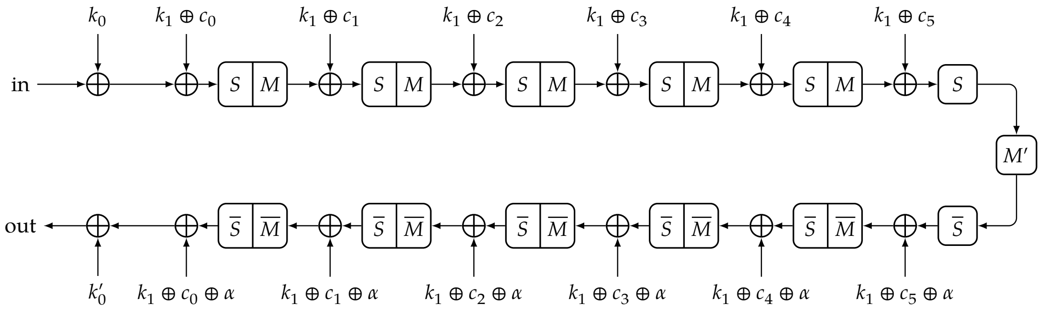
PRINCE is a great cipher.
$\texttt{PRINCE}_k(p) = c$ with 64 bit texts and 128 bit key.

Optimized with respect to latency when implemented in hardware.

It looks like an onion:
Symmetric structure around an involutory double round, wrapped in FX construction:
Can reuse encryption circuit for decryption.

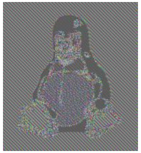
Can encrypt/decrypt in very short time: admits single clock operation for good frequencies, can be implemented for 8ns latency (few clock cycles added latency even for very fast memory interfaces).

Here comes the PRINCE


k_0
$k_1 \oplus c_0$
$k_1 \oplus c_1$
$k_1 \oplus c_2$
$k_1 \oplus c_3$
$k_1 \oplus c_4$
$k_1 \oplus c_5$
in
S
M
M'
out
$\overline{S}$
$\overline{M}$
k_0'
$k_1 \oplus c_0 \oplus \alpha$
$k_1 \oplus c_1 \oplus \alpha$
$k_1 \oplus c_2 \oplus \alpha$
$k_1 \oplus c_3 \oplus \alpha$
$k_1 \oplus c_4 \oplus \alpha$
$k_1 \oplus c_5 \oplus \alpha$

Uses

- Direct encryption, ECB mode: just encrypt each block, but penguins remain penguins.



- Use a XEX-like mode:

$$\text{encrypted line} = W \oplus \mathtt{PRINCE}_k(\text{clear line} \oplus W)$$

where W is securely derived from an IV, (optional: version counter) and address line
Uh: Need to generate only 64 bits from all this information. Good function $\Rightarrow$ more latency.
Some solutions use PRINCE itself (and/or Galois Multiplication) to generate the PRINCE whitening and core keys by encrypting version number and address using other keys.
$\Rightarrow$ latency becomes 2 or 3 times higher.

- Use a counter mode:

$$\text{encrypted line} = \text{clear line} \oplus \mathtt{PRINCE}_k(\text{IV} \| \text{address}\, \{\|\text{version counter}\})$$

Uh: Again need to compress the info to just 64 bits before using PRINCE. Same problem.

Idea

Use a *tweaked* cipher:

$$c = \mathtt{TWEAKED\text{-}PRINCE}_{K,T}(m)$$

with 64-bit texts and tweak, 128-bit key.

For applications without a version counter, and per-security domain key, just use the address as tweak.

If you need a version counter, you have to compress less info (or try to use a larger tweak).

In all cases - less latency added.

A 64-bit tweak is ok (for the moment) because of the actual use cases.

Ciphers that help against software exploitation 1/4

Deja vu from LCD 2014 and LCD 2015.

Turn tables around! Use crypto to harden software security: *Code Flow Integrity*).

Nathan Tuck, Brad Calder, and George Varghese, 37th IEEE/ACM MICRO, 2004.
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 with full AES as round function. Sure.)

Ali José Mashtizadeh, Andrea Bittau, David Mazières, Dan Boneh
Cryptographically Enforced Control Flow Integrity (`http://arxiv.org/abs/1408.1451`)

Compute AES on each pointer, store result separately as a 128-bit MAC, store key schedule in 11 XMM registers, use AES-NI. Alternative: a SHA-256 based HMAC.

(Still about 40 cycles per MAC calculation/verification.)

Ciphers that help against software exploitation 1/4

Deja vu from LCD 2014 and LCD 2015.

Turn tables around! Use crypto to harden software security: *Code Flow Integrity*).

Nathan Tuck, Brad Calder, and George Varghese, 37th IEEE/ACM MICRO, 2004.
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 with full AES as round function. Sure.)

Ali José Mashtizadeh, Andrea Bittau, David Mazières, Dan Boneh
Cryptographically Enforced Control Flow Integrity (`http://arxiv.org/abs/1408.1451`)

Compute AES on each pointer, store result separately as a 128-bit MAC, store key schedule in 11 XMM registers, use AES-NI. Alternative: a SHA-256 based HMAC.

(Still about 40 cycles per MAC calculation/verification.)

Ciphers that help against software exploitation 1/4

Deja vu from LCD 2014 and LCD 2015.
Turn tables around! Use crypto to harden software security: *Code Flow Integrity*).

Nathan Tuck, Brad Calder, and George Varghese, 37th IEEE/ACM MICRO, 2004.
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 with full AES as round function. Sure.)

Ali José Mashtizadeh, Andrea Bittau, David Mazières, Dan Boneh
Cryptographically Enforced Control Flow Integrity (`http://arxiv.org/abs/1408.1451`)

Compute AES on each pointer, store result separately as a 128-bit MAC, store key schedule in 11 XMM registers, use AES-NI. Alternative: a SHA-256 based HMAC.

(Still about 40 cycles per MAC calculation/verification.)

Ciphers that help against software exploitation 1/4

Deja vu from LCD 2014 and LCD 2015.

Turn tables around! Use crypto to harden software security: *Code Flow Integrity*).

Nathan Tuck, Brad Calder, and George Varghese, 37th IEEE/ACM MICRO, 2004.

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 with full AES as round function. Sure.)

Ali José Mashtizadeh, Andrea Bittau, David Mazières, Dan Boneh

Cryptographically Enforced Control Flow Integrity (`http://arxiv.org/abs/1408.1451`)

Compute AES on each pointer, store result separately as a 128-bit MAC, store key schedule in 11 XMM registers, use AES-NI. Alternative: a SHA-256 based HMAC.

(Still about 40 cycles per MAC calculation/verification.)

Ciphers that help against software exploitation 1/4

Deja vu from LCD 2014 and LCD 2015.
Turn tables around! Use crypto to harden software security: *Code Flow Integrity*).

Nathan Tuck, Brad Calder, and George Varghese, 37th IEEE/ACM MICRO, 2004.
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 with full AES as round function. Sure.)

Ali José Mashtizadeh, Andrea Bittau, David Mazières, Dan Boneh
Cryptographically Enforced Control Flow Integrity (`http://arxiv.org/abs/1408.1451`)

Compute AES on each pointer, store result separately as a 128-bit MAC, store key schedule in 11 XMM registers, use AES-NI. Alternative: a SHA-256 based HMAC.

(Still about 40 cycles per MAC calculation/verification.)

Ciphers that help against software exploitation 1/4

Deja vu from LCD 2014 and LCD 2015.
Turn tables around! Use crypto to harden software security: *Code Flow Integrity*).

Nathan Tuck, Brad Calder, and George Varghese, 37th IEEE/ACM MICRO, 2004.
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 with full AES as round function. Sure.)

Ali José Mashtizadeh, Andrea Bittau, David Mazières, Dan Boneh
Cryptographically Enforced Control Flow Integrity (`http://arxiv.org/abs/1408.1451`)

Compute AES on each pointer, store result separately as a 128-bit MAC, store key schedule in 11 XMM registers, use AES-NI. Alternative: a SHA-256 based HMAC.

(Still about 40 cycles per MAC calculation/verification.)

Ciphers that help against software exploitation 2/4

We need something better - and now comes the FUN part.

Observation: we do not need to encrypt the whole addresses.

We just need to *verify* these addresses in the code itself (google CFI, LLVM CFI, Boneh does it as well).

But why waste 128 bits per pointer?
Why use an expensive function such as AES or SHA-256?

Now, let us insert a (new, lightweigth) keyed and tweaked MAC for pointer addresses in unused bits of address space (something like 6 or 7 bits)

Ciphers that help against software exploitation 2/4

We need something better - and now comes the FUN part.

Observation: we do not need to encrypt the whole addresses.

We just need to *verify* these addresses in the code itself (google CFI, LLVM CFI, Boneh does it as well).

But why waste 128 bits per pointer?

Why use an expensive function such as AES or SHA-256?

Now, let us insert a (new, lightweigth) keyed and tweaked MAC for pointer addresses in unused bits of address space (something like 6 or 7 bits)

Ciphers that help against software exploitation 2/4

We need something better - and now comes the FUN part.

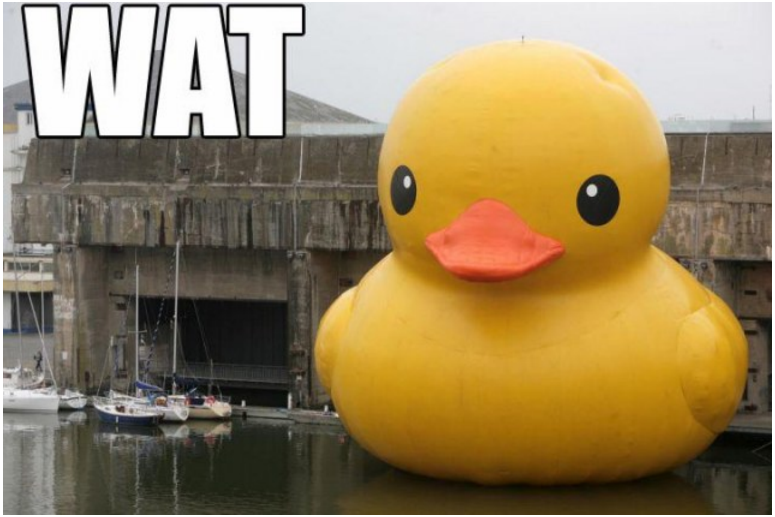
Observation: we do not need to encrypt the whole addresses.

We just need to *verify* these addresses in the code itself (google CFI, LLVM CFI, Boneh does it as well).

But why waste 128 bits per pointer?
Why use an expensive function such as AES or SHA-256?

Now, let us insert a (new, lightweigth) keyed and tweaked MAC for pointer addresses in unused bits of address space (something like 6 or 7 bits)

Ciphers that help against software exploitation 3/4



Ciphers that help against software exploitation 4/4

In ROP you return from functions you did not enter properly

Function prologue

```
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

```
}
```

- 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.
Computed using a (compression function based on a) tweaked block cipher, truncated.

Remark: Some higher execution level controls the key.

Ciphers that help against software exploitation 4/4

In ROP you return from functions you did not enter properly

Function prologue

```
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

```
}
```

- 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.
Computed using a (compression function based on a) tweaked block cipher, truncated.

Remark: Some higher execution level controls the key.

The road towards QARMA

Start with the PRINCE design.

Add a tweak: just add the tweak to each round.

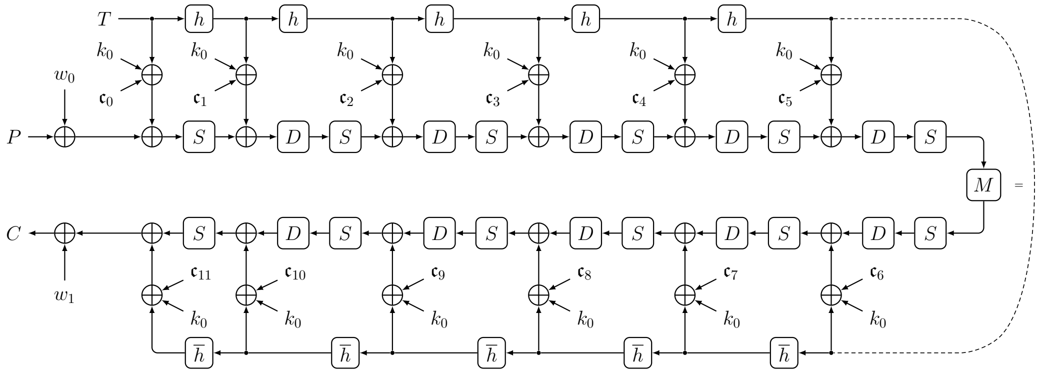
First result: MANTIS. PRINCE latency with two extra rounds in order to guarantee sufficient tweak diffusion and resistance against related tweak attacks.

However, in order to minimise also gate counts, rounds from MIDORI recycled, which may be problematic with acceptance (more later).

Note that total area in adding circuits to support software security or memory encryption, and therefore the power consumption, is minimal wrt rest of core using the features. 10%-20% more area, as long as latency still similar, is perfectly ok.

Also, targeting only fully unrolled and maybe pipelined implementations, so we gain nothing from restricting to involutory elements. This leads to QARMA.

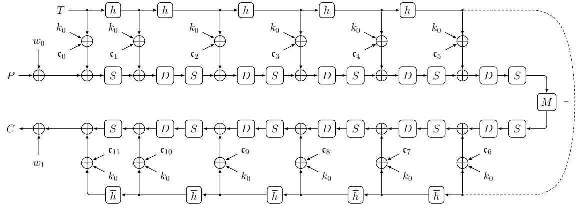
MANTIS



D = Shuffle of the Cells followed by multiplication by circ$(0, 1, 1, 1)$

(author list omitted, with some inspiration/brainstorming with Roberto Avanzi)

MANTIS



1. Recycle MIDORI round (S-Box, internal state permutation, diffusion matrix)
2. Use round constants that add to all bits
3. S-Box has 4 fixed points
4. Almost MDS matrix circ$(0, 1, 1, 1)$
5. Tweak cells are shuffled (h) between each round
6. Tweak permutation is determined using MILP
7. In related tweak setting, $r = 6$ means 44 active S-Boxes, $r = 7$ implies 50
8. However, S-Box and diffusion layer not seen well by industrial community

Beyond MANTIS

One can agree or not with the assessment that reusing some MIDORI components may be dangerous. But this is something one has to take into account with real world requirements (and it should be, after all, relatively easy to describe keys that allow to “peel” one round in the core cipher).

So we do the following: We try to recycle the security proofs as much as possible and build in better security margins. Boring job, try to spice it by doing some mathematics.

1. Use better S-Boxes - develop very fast heuristic methods to find low latency ones (new!)

   And allow use of the PRINCE one to make people happy
2. Create diffusion matrix that adds rotations (new!)
3. Modify reflector with keying to make it non-involutory: pseudo-reflector
4. Replace FX with single key 3-round EM that collapses to 2-round EM (instead of 1-round) if large class of fixed self-differentials of pseudo-reflector hit
5. Oh, yes and meditate on a 128 bit variant as well (new!)

Beyond MANTIS

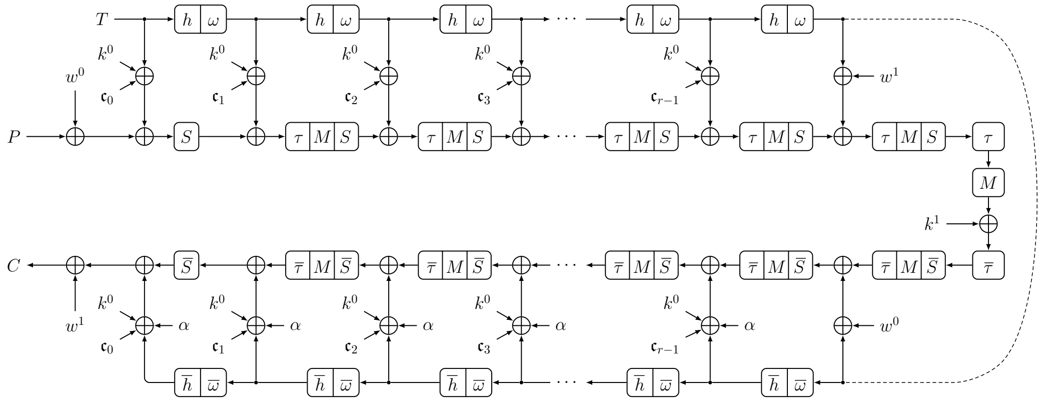
One can agree or not with the assessment that reusing some MIDORI components may be dangerous. But this is something one has to take into account with real world requirements (and it should be, after all, relatively easy to describe keys that allow to "peel" one round in the core cipher).

So we do the following: We try to recycle the security proofs as much as possible and build in better security margins. Boring job, try to spice it by doing some mathematics.

1. Use better S-Boxes - develop very fast heuristic methods to find low latency ones (new!)

   And allow use of the PRINCE one to make people happy
2. Create diffusion matrix that adds rotations (new!)
3. Modify reflector with keying to make it non-involutory: pseudo-reflector
4. Replace FX with single key 3-round EM that collapses to 2-round EM (instead of 1-round) if large class of fixed self-differentials of pseudo-reflector hit
5. Oh, yes and meditate on a 128 bit variant as well (new!)

QARMA



τ, h = Shuffles of the cells, M = multiplication by an Almost MDS matrix
S = S-Box layer, ω = LSFR on 7/16 cells to keep non-zero values non-constant

Almost MDS Matrix over $R = \mathbb{F}_2[X]/(X^m + 1) = R[\rho]$

Represent state as a matrix of sixteen m-bit cells:

$$\mathrm{IS} = \begin{pmatrix} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ s_{12} & s_{13} & s_{14} & s_{15} \end{pmatrix} .$$

Let $R = \mathbb{F}_2[\rho]$ be the quotient ring $\mathbb{F}_2[X]/(X^m + 1)$ where ρ is the image of X in R. $\rho^m = 1$.
Consider matrices that operates on columns (i.e. on the left) of form

$$M = \mathrm{circ}(0, \rho^a, \rho^b, \rho^c) = \begin{pmatrix} 0 & \rho^a & \rho^b & \rho^c \\ \rho^c & 0 & \rho^a & \rho^b \\ \rho^b & \rho^c & 0 & \rho^a \\ \rho^a & \rho^b & \rho^c & 0 \end{pmatrix} . \qquad (*)$$

Note that ρ^a means circular rotation of a places to the left.
Remark: Multiplication by ρ^a in $\mathbb{F}[\rho]$ is cheaper than any multiplication in $\mathbb{F}_{2^m}$ except by 0 or 1.
In fact, these matrices are as expensive (area, latency) as the $\{0, 1\}$-matrices (well, some wire crossing).

Classification

Theorem 1. *Let $R = \mathbb{F}_2[\rho]$ be the quotient ring $\mathbb{F}_2[X]/(X^m+1)$ where ρ is the image of X in R, $m \geqslant 2$. The Matrices M of the form $\mathrm{circ}(0, \rho^a, \rho^b, \rho^c)$ over the ring R that are Almost MDS are precisely the invertible ones, i.e. are those for which $\rho^{4a} + \rho^{4b} + \rho^{4c} \in R^*$.*

$$\begin{pmatrix} 0 & \rho^a & \rho^b & \rho^c \\ \rho^c & 0 & \rho^a & \rho^b \\ \rho^b & \rho^c & 0 & \rho^a \\ \rho^a & \rho^b & \rho^c & 0 \end{pmatrix}$$

(For $m = 4$, this is $1 + 1 + 1 = 1$. For $m = 8$ this is either 1 or ρ^4. So always invertible.)

Theorem 2. *Let $R = \mathbb{F}_2[X]/(X^m + 1) = \mathbb{F}_2[\rho]$ be defined as in Theorem 1.*
The Almost MDS matrices $M = \mathrm{circ}(0, \rho^a, \rho^b, \rho^c)$ which admit an inverse of the same form $\overline{M} = \mathrm{circ}(0, \rho^d, \rho^e, \rho^f)$, i.e. with entries of weight at most one, are those that satisfy $a \equiv c + \mu$ where $2\,\mu \equiv 0 \bmod m$ (for odd m this implies $\mu = 0$, for even m it can be $\mu = 0$ or $m/2$).

In this case, the parameters of the matrix $\overline{M}$ are: $d \equiv a - 2\,b$, $e \equiv -b$, and $f \equiv d + \mu \bmod m$.

The involutory matrices M are those for which, additionally, $2\,b \equiv 0$.

Classification

Theorem 2. *$M = \text{circ}(0, \rho^a, \rho^b, \rho^c)$ with inverse $\overline{M} = \text{circ}(0, \rho^d, \rho^e, \rho^f)$, implies $a \equiv c + \mu$ where $2\mu \equiv 0 \bmod m$; $d \equiv a - 2b$, $e \equiv -b$, and $f \equiv d + \mu \bmod m$.*

The involutory matrices M are those for which, additionally, $2b \equiv 0$.

$$\begin{pmatrix} 0 & \rho^a & \rho^b & \rho^c \\ \rho^c & 0 & \rho^a & \rho^b \\ \rho^b & \rho^c & 0 & \rho^a \\ \rho^a & \rho^b & \rho^c & 0 \end{pmatrix}$$

Hence, *there are two degrees of freedom to define the matrices M, and one extra bit in case m is even. If M has to be involutory, there is only one degree of freedom (and two bits).*
Note that, for fixed a, b, and c, all matrices $M = \text{circ}(0, \rho^{a+k}, \rho^{b+k}, \rho^{c+k})$ form a class with the same output up to circular rotation of all cells by the same amount, and in each such class there is always one (or m odd) or two (for m even) elements which are involutory.

Choice

Ideally pick $M = \mathrm{circ}(0, \rho^a, \rho^b, \rho^c)$ such that all $x - y$ with $x \neq y \in \{a, b, c\}$ different mod m, so that common sums of two S-Box outputs in same result column have different relative shifts.

Only partially possible for $m = 8$, not for $m = 4$. Examples with $m = 8$:

$$\begin{pmatrix} 0 & \rho^1 & \rho^4 & \rho^5 \\ \rho^5 & 0 & \rho^1 & \rho^4 \\ \rho^4 & \rho^5 & 0 & \rho^1 \\ \rho^1 & \rho^4 & \rho^5 & 0 \end{pmatrix} \times \begin{pmatrix} v_0 \\ v_1 \\ \cdots \\ \cdots \end{pmatrix} = \begin{pmatrix} (v_1 \lll 1) \oplus \cdots \\ (v_0 \lll 5) \oplus \cdots \\ \boxed{(v_0 \lll 4) \oplus (v_1 \lll 5)} \oplus \cdots \\ \boxed{(v_0 \lll 1) \oplus (v_1 \lll 4)} \oplus \cdots \end{pmatrix} \begin{matrix} \\ \\ \Delta = 1 \\ \Delta = 3 \end{matrix}$$

But at least common rotation different...

And the common (joint) amounts are always different. Then next S-Box layer more likely to disrupt characteristics (linear, differential, also higher order, including saturation), or at least to avoid it to be copied onto two output boxes (problem with $\{0, 1\}$-matrices).

Select values heuristically, minimising fixed points (also under simple linear transformations) over 1 or 1.5 rounds - should redo with some characteristics.

Choice

Ideally pick $M = \mathrm{circ}(0, \rho^a, \rho^b, \rho^c)$ such that all $x - y$ with $x \neq y \in \{a, b, c\}$ different mod m, so that common sums of two S-Box outputs in same result column have different relative shifts.

Only partially possible for $m = 8$, not for $m = 4$. Examples with $m = 8$:

$$\begin{pmatrix} 0 & \rho^1 & \rho^4 & \rho^5 \\ \rho^5 & 0 & \rho^1 & \rho^4 \\ \rho^4 & \rho^5 & 0 & \rho^1 \\ \rho^1 & \rho^4 & \rho^5 & 0 \end{pmatrix} \times \begin{pmatrix} v_0 \\ \cdots \\ v_2 \\ \cdots \end{pmatrix} = \begin{pmatrix} (v_2 \lll 4) \oplus \cdots \\ \boxed{(v_0 \lll 5) \oplus (v_2 \lll 1)} \oplus \cdots \\ (v_0 \lll 4) \oplus \cdots \\ \boxed{(v_0 \lll 1) \oplus (v_2 \lll 5)} \oplus \cdots \end{pmatrix} \begin{matrix} \\ \Delta = 4 \\ \\ \Delta = 4 \end{matrix}$$

But at least common rotation different…

And the common (joint) amounts are always different. Then next S-Box layer more likely to disrupt characteristics (linear, differential, also higher order, including saturation), or at least to avoid it to be copied onto two output boxes (problem with $\{0, 1\}$-matrices).

Select values heuristically, minimising fixed points (also under simple linear transformations) over 1 or 1.5 rounds - should redo with some characteristics.

Choice

Ideally pick $M = \mathrm{circ}(0, \rho^a, \rho^b, \rho^c)$ such that all $x - y$ with $x \neq y \in \{a, b, c\}$ different mod m, so that common sums of two S-Box outputs in same result column have different relative shifts.

Only partially possible for $m = 8$, not for $m = 4$. Examples with $m = 8$:

$$\begin{pmatrix} 0 & \rho^1 & \rho^4 & \rho^5 \\ \rho^5 & 0 & \rho^1 & \rho^4 \\ \rho^4 & \rho^5 & 0 & \rho^1 \\ \rho^1 & \rho^4 & \rho^5 & 0 \end{pmatrix} \times \begin{pmatrix} v_0 \\ \cdots \\ v_2 \\ \cdots \end{pmatrix} = \begin{pmatrix} (v_2 \lll 4) \oplus \cdots \\ \boxed{(v_0 \lll 5) \oplus (v_2 \lll 1)} \oplus \cdots \\ (v_0 \lll 4) \oplus \cdots \\ \boxed{(v_0 \lll 1) \oplus (v_2 \lll 5)} \oplus \cdots \end{pmatrix} \begin{matrix} \\ \Delta = 4 \\ \\ \Delta = 4 \end{matrix}$$

But at least common rotation different...
And the common (joint) amounts are always different. Then next S-Box layer more likely to disrupt characteristics (linear, differential, also higher order, including saturation), or at least to avoid it to be copied onto two output boxes (problem with $\{0, 1\}$-matrices).
Select values heuristically, minimising fixed points (also under simple linear transformations) over 1 or 1.5 rounds - should redo with some characteristics.

Remarks

Each output is the XOR of three values

$$\begin{pmatrix} 0 & \rho^a & \rho^b & \rho^c \\ \rho^c & 0 & \rho^a & \rho^b \\ \rho^b & \rho^c & 0 & \rho^a \\ \rho^a & \rho^b & \rho^c & 0 \end{pmatrix}$$

NAND3, NOR3 can be optimised,
but XOR3/XNOR3 are a mess

So you have to cascade two XORs anyway

So we could allow a total weight 4 per column in both matrix and inverse: More weight, but same latency

$$\begin{pmatrix} 0 & \rho^a & \rho^b + \rho^{b'} & \rho^c \\ \rho^c & 0 & \rho^a & \rho^b + \rho^{b'} \\ \rho^b + \rho^{b'} & \rho^c & 0 & \rho^a \\ \rho^a & \rho^b + \rho^{b'} & \rho^c & 0 \end{pmatrix} ?$$

Do these matrices exist?
Can we do MDS? (Impossible with $\{0, 1\}$)

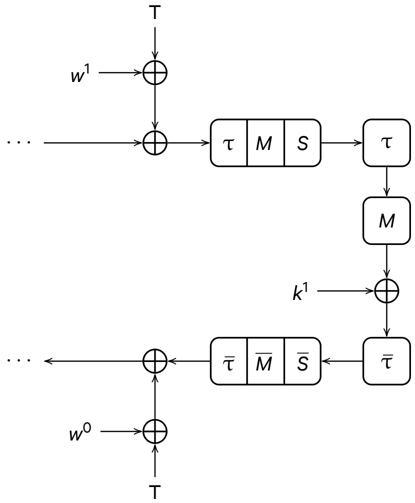
$$\begin{pmatrix} 1 & \rho^a & \rho^b & \rho^c \\ \rho^c & 1 & \rho^a & \rho^b \\ \rho^b & \rho^c & 1 & \rho^a \\ \rho^a & \rho^b & \rho^c & 1 \end{pmatrix} ?$$

Open questions

New Central Construction

Central rounds have a few important differences:

- Use whitening key(s) instead of core key
  - Use PRINCE whitening key expansion
  - w^0 is 1-to-1 with w^1 and 1-to-1 with $w^0 \oplus w^1$
  - Makes reflection attacks more difficult
- Non involutory *Pseudo-Reflector*
  - Add key (not tweak) to continue core key mixing
  - Becomes $M \cdot k^1$ for decryption
  - Non involutory in general, but it is for some special k^1
  - Also makes reflection attacks more difficult
- For `QARMA-128`, M has $2^{72}/2^{128}$ fixed points
  - Not ideal as square root …
    … but better than 2^{96} as with $\{0,1\}$-matrices
  - `QARMA-64`, as `MIDORI`/`MANTIS`, still has $2^{48}/2^{64}$
    … but better against differential characteristics in other places of the cipher



S-Box Search Heuristics

Most important characteristic of a (cryptographically good) S-Box: total latency

Logic synthesis of a circuit is expensive (for instance, time consuming)

Requires expensive tools that are also big and slow: you do not want to batch operate them on billions of inputs

Doing that once on all cryptographically ok S-Boxes and store in a huge database not practical (Not even "practical" in the cryptanalytic sense that you "only" need $2^{62.17}$ entries)

Hence we do a brutal first sieving that ultimately proved much more effective than hoped

S-Box Search Heuristics

1. Loop over the class of S-Boxes you want to sieve
   - (Such as all involutory ones using Prissette's algorithm)
   - (Or all linearly equivalent to some special one)
2. Use Quine-McCluskey algorithm to compute the SOP and NOT-SOP of each output but of the S-Box (by De Morgan's law, the second gives the right result upon negation of all inputs and swapping NAND with NOR)
3. Compute the *depth* of each bit (and its negation) using also 3-input NAND and NOR gates (which are 1.5 times deep than a 2-input NAND and NOR), keep the minimum
4. Take the maximum over all output bits
5. Keep S-Boxes for which this value is under a set threshold

Note: depth almost always matching best implementations in literature, rarely only 0.5 GE more

Use logic synthesis later to get smaller area, but difference not huge

S-Box Search Heuristics

Example: the MIDORI Sb_0 S-Box:

```
SBOX = [ 12, 10, 13, 3, 14, 11, 15, 7, 8, 9, 1, 5, 0, 2, 4, 6 ]
SINV = [ 12, 10, 13, 3, 14, 11, 15, 7, 8, 9, 1, 5, 0, 2, 4, 6 ]

fixed points 4, max arith diff 3, max bitwise diff 4, linearity 8
flipping bit 0 in input, counts for 0..DEG flipped bits = 0 10 4 2 0
flipping bit 1 in input, counts for 0..DEG flipped bits = 0 8 8 0 0
flipping bit 2 in input, counts for 0..DEG flipped bits = 0 10 4 2 0
flipping bit 3 in input, counts for 0..DEG flipped bits = 0 4 8 4 0

QMC of SBOX
0011011101110000 -- Result: wyz' + wy'z + xz' + xy'
0101111100000101 -- Result: wz' + yz' + wy
1010101100010011 -- Result: wxz + w'z' + xy
1110111011000000 -- Result: x'z' + x'y' + w'z'
QMC of not SBOX
1100100001001111 -- Result: x'y'z' + w'x' + yz
1010000011111010 -- Result: w'y' + y'z + w'z
0101010011101100 -- Result: wy'z' + w'y'z + wx' + x'z
0001000100111111 -- Result: wx + xz + yz
```

S-Box Search Heuristics

Extreme example: NLFSR

$$\text{out} = w \oplus x \oplus (\overline{x} \wedge \overline{y} \wedge \overline{z})$$

Critical path has depth 4 (2 XORs).

QMC gives you

$$(\overline{x} \wedge \overline{y} \wedge \overline{z}) \vee (\overline{w} \wedge x \wedge z) \vee (\overline{w} \wedge y \wedge z) \vee (w \wedge \overline{z})$$

NOT layer = depth 0.5
NAND/NOR dominates next layer: depth 1.5
OR of four elements (actually a NAND since you negate the previous layer): depth 2

Total 4 (difference in area big: 5.5 GE vs 8.5 GE)
(Creepy)

S-Box Search Heuristics

Extreme example: NLFSR

$$\text{out} = w \oplus x \oplus (\overline{x} \wedge \overline{y} \wedge \overline{z})$$

Critical path has depth 4 (2 XORs).

QMC gives you

$$(\overline{x} \wedge \overline{y} \wedge \overline{z}) \vee (\overline{w} \wedge x \wedge z) \vee (\overline{w} \wedge y \wedge z) \vee (w \wedge \overline{z})$$

NOT layer = depth 0.5

NAND/NOR dominates next layer: depth 1.5

OR of four elements (actually a NAND since you negate the previous layer): depth 2

Total 4 (difference in area big: 5.5 GE vs 8.5 GE)

(Creepy)

S-Box Search Heuristics

Extreme example: NLFSR

$$\text{out} = w \oplus x \oplus (\overline{x} \wedge \overline{y} \wedge \overline{z})$$

Critical path has depth 4 (2 XORs).

QMC gives you

$$(\overline{x} \wedge \overline{y} \wedge \overline{z}) \vee (\overline{w} \wedge x \wedge z) \vee (\overline{w} \wedge y \wedge z) \vee (w \wedge \overline{z})$$

NOT layer = depth 0.5
NAND/NOR dominates next layer: depth 1.5
OR of four elements (actually a NAND since you negate the previous layer): depth 2

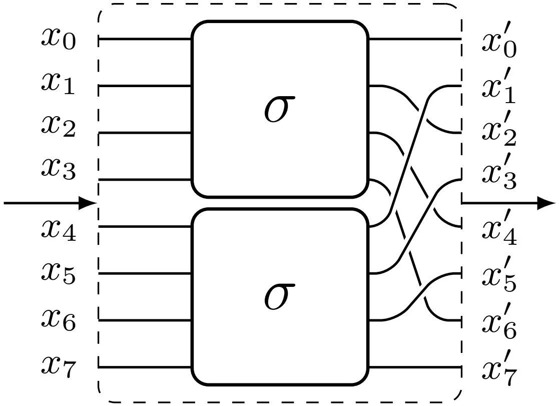
Total 4 (difference in area big: 5.5 GE vs 8.5 GE)
(Creepy)

S-Box Search Heuristics

Three S-Boxes selected

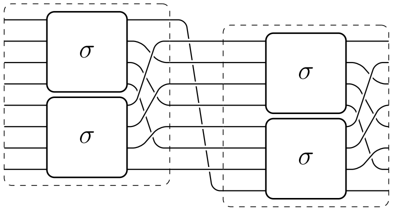
1. One with same parameters and depth as MIDORI's, but slightly larger area, only 2 fixed points (curiosity: all such S-Boxes have 2 fixed points whose values have Hamming distance 1).
2. Similar one but such that one bit input differences always give same number of n-bit output differences for each $n = 1, 2, 3, 4$ – for each fixed input bit.
3. The original PRINCE S-Box (but our implementation reduces depth of S-box and inverse from 5 and 5 to 4 and 4.5)

The 8-bit S-Box

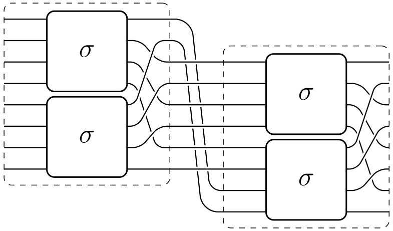


The 8-bit S-Box

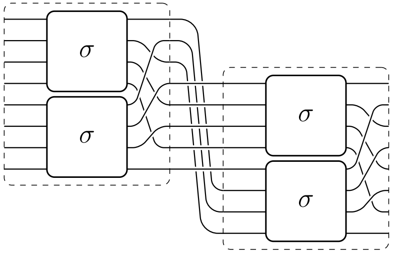
1:



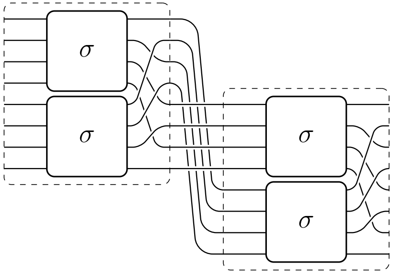
2:



3:



4:



Implementation

We consider here gate depth (and to a lesser extent, area), and only the 64 bit version.

σ_0 is the MIDORI-like S-Box, σ_2 is the PRINCE S-Box.

Values are estimates.

Just as rule of thumb: 128 bit version has roughly $2r'/r$ the area, r'/r the depth as QARMA-64.

Cipher	Depth (GE)	Area (GE)	Latency (ns)
QARMA$_5$-σ_0	100	8682	15.25
QARMA$_6$-σ_0	117	9978	17.90
QARMA$_7$-σ_0	134	11211	19.85
QARMA$_5$-σ_2	107	9414	16.8
QARMA$_6$-σ_2	125	10822	19.2
QARMA$_7$-σ_2	143	12160	21.5
MANTIS$_5$	100	8304	15.22
MANTIS$_6$	117	9559	17.90
MANTIS$_7$	134	10753	19.83
PRINCE	114	8263	17.50

For QARMA-128 the “central” value of r' would be 9. Hence consider $r' = 8$, 9, and 10.

Outline

Conclusion

- Real World Crypto means you have to deal with Real World Constraints
- (Also RWC. Not a coincidence!)
- Prevention is the best medicine
- We considered how to improve upon and at the same time keep close to existing, established designs with proofs against generic attacks for specific applications
- First result: New class of Almost MDS matrices, better disruption of iterative characteristics, improved reflectors
- Second result: Fast maximal latency S-Box sieving heuristics that proved even better than initially hoped
- Third result: 3-round EM construction around non-involutory pseudo-reflector against reflection attacks, attacks on EM with an involution, slide(x) with a twist, etc…
- More ideas …

תודה