Permutation-based cryptography for the Internet of Things

Gilles VAN ASSCHE\textsuperscript{1}

Joint work with Guido BERTONI, Joan DAEMEN\textsuperscript{1,2}, Seth HOFFERT, Michaël PEETERS\textsuperscript{1} and Ronny VAN KEER\textsuperscript{1}

\textsuperscript{1}STMicroelectronics
\textsuperscript{2}Radboud University

RIOT Summit 2017
Berlin, September 25-26, 2017
Outline

1. Parameters for the IoT
2. Permutations!
3. Keyed applications
4. STROBE
5. KETJE and KEYAK
6. KRAVATTE and the Farfalle construction
Outline

1 Parameters for the IoT
2 Permutations!
3 Keyed applications
4 STROBE
5 KETJE and KEYAK
6 KRAVATTE and the Farfalle construction
On the cost of cryptography for the IoT

- code size
- memory usage
- execution time
- efficiency on the high-end server?
- protections against side-channel attacks?
On the cost of cryptography for the IoT

- code size
- memory usage
- execution time
- efficiency on the high-end server?
- protections against side-channel attacks?
On the cost of cryptography for the IoT

- code size
- memory usage
- execution time
- efficiency on the high-end server?
- protections against side-channel attacks?
What are side-channel attacks?

- Leakage from the device
  - Time, electrical consumption, EM radiation
  - *simple power analysis (SPA)* vs *differential power analysis (DPA)*
What are side-channel attacks?

- Inducing faults in the device
  - Glitch, laser pulse

![Picture by ViaMoi on Flickr](image.png)
Usage and ownership

Actors:
- Key owner
- Device owner
- Actual user

Usually, these are the same person, but...
Usage and ownership

When key owner \( \neq \) device owner

- Banking card
- DRM

But hopefully the same person in open-source contexts!
Usage and ownership

When key/device owner ≠ actual user

- Not always controlling the device
  - E.g., devices spread over a large area
  - E.g., on-site personnel
  - E.g., lost device

- Distant eavesdropping

Protections against SCA can be needed.
Outline

1. Parameters for the IoT
2. Permutations!
3. Keyed applications
4. STROBE
5. KETJE and KEYAK
6. KRAVATTE and the Farfalle construction
Symmetric crypto: what textbooks and intro’s say

Symmetric cryptographic primitives:
- Block ciphers
- Stream ciphers
- Hash functions
And their modes-of-use
Examples of permutations

- In Salsa, Chacha, Grindhal...
- In SHA-3 candidates: CubeHash, Grøstl, JH, MD6, ...
- In CAESAR candidates: Ascon, Icepole, Norx, π-cipher, Primates, Stribob, ...

And of course in KECCAK
The sponge construction

- Calls a permutation $f$
- The capacity $c$ determines the generic security:
  - Hashing: $2^{c/2}$
  - Authentication, encryption: $2^{c-\epsilon}$
The seven permutation army:
- 25, 50, 100, 200, 400, 800, 1600 bits
- toy, lightweight, fastest
- standardized in [FIPS 202]

Repetition of a simple round function
- that operates on a 3D state
- \((5 \times 5)\) lanes
- up to 64-bit each
KECCAK-f in pseudo-code

KECCAK-f[b](A) {
  forall i in 0...n_r-1
    A = Round[b](A, RC[i])
  return A
}

Round[b](A,RC) {
  \theta step
  C[x] = A[x,0] \ xor A[x,1] \ xor A[x,2] \ xor A[x,3] \ xor A[x,4], forall x in 0...4
  D[x] = C[x-1] \ xor rot(C[x+1],1),
  A[x,y] = A[x,y] \ xor D[x],

  \rho and \pi steps
  B[y,2*x+3*y] = rot(A[x,y], r[x,y]),

  \chi step
  A[x,y] = B[x,y] \ xor ((not B[x+1,y]) and B[x+2,y]),

  \iota step
  A[0,0] = A[0,0] \ xor RC

  return A
}

https://keccak.team/keccak_specs_summary.html
Bit interleaving

\[ \text{ROT}_{64} \leftrightarrow 2 \times \text{ROT}_{32} \]
The unbearable lightness of permutations

- Example: hashing with target security strength $2^{c/2}$
  - Davies-Meyer block cipher based hash
    - chaining value (block size): $n \geq c$
    - input block size ("key" length): typically $k \geq n$
    - feedforward (block size): $n$
    - $\Rightarrow$ total state $\geq 3c$
  - Sponge
    - permutation width: $c + r$
    - $r$ can be made arbitrarily small, e.g., 1 byte
    - $\Rightarrow$ total state $\geq c + 8$
Cost of primitives and modes together

- Our multi-purpose Keccak outperforms our multi-purpose AES in terms of throughput over area by an average of 4.0.
- Typically a *plain* AES is much smaller than a *plain* Keccak.
- Addition of modes is more costly for AES than Keccak
  ⇒ Keccak is more flexible than AES.
Symmetric crypto: a more correct picture

Symmetric cryptographic primitives:

- Block ciphers
- Key stream generators
- **Permutations**

And their modes-of-use

Picture by Sébastien Wiertz
Keyed applications

Outline

1. Parameters for the IoT
2. Permutations!
3. Keyed applications
4. STROBE
5. KETJE and KEYAK
6. KRAVATTE and the Farfalle construction
Use Sponge for MACing
Use Sponge for (stream) encryption
Single pass authenticated encryption

- Key
- IV
- Padded message
- Key stream
- MAC

But this is no longer the sponge ...
The duplex construction

- Generic security provably equivalent to that of sponge
- Applications: authenticated encryption, reseedable pseudorandom generator ...
Outline

1. Parameters for the IoT
2. Permutations!
3. Keyed applications
4. Strobe
5. Ketje and Keyak
6. Kravatte and the Farfalle construction
What is Strobe?

- Layer above the duplex construction
- Safe and easy syntax, to achieve, e.g.,
  - secure channels
  - signatures over a complete session
- Very compact implementation
- Mechanism to prevent side-channel attacks

[Mike Hamburg — https://strobe.sourceforge.io/]
# Operations and data flow in STROBE

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Operation</th>
<th>Flags</th>
<th>Application</th>
<th>STROBE</th>
<th>Transport</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY</td>
<td>Secret key</td>
<td>AC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD</td>
<td>Associated data</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRF</td>
<td>Hash / PRF</td>
<td>IAC</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>CLR</td>
<td>Send cleartext data</td>
<td>A T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>recv-CLR</td>
<td>Receive cleartext data</td>
<td>IA T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENC</td>
<td>Encrypt</td>
<td>ACT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>recv-ENC</td>
<td>Decrypt</td>
<td>IACT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC</td>
<td>Compute MAC</td>
<td>CT</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>recv-ENC</td>
<td>Verify MAC</td>
<td>I CT</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>RATCHET</td>
<td>Rekey to prevent rollback</td>
<td>C</td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Legend:  
- □ Send/recv  
- ○ Absorb into sponge  
- ⊕ Xor with cipher  
- ⊙ Roll key  

Figure courtesy of Mike Hamburg
Example: key derivation

- **KEY**(master shared key $K$)
- **RATCHET**
- derived key 1 $\leftarrow$ **PRF**(16 bytes)
- **RATCHET**
- derived key 2 $\leftarrow$ **PRF**(16 bytes)
Example: protocol

- **KEY**(shared key \( K \))
- **AD**[nonce](sequence number \( i \))
- **AD**[auth-data](client IP address | server IP address)
- **send_ENC**(“GET file”)
- **send_MAC**(128 bits)
- **recv_ENC**(buffer)
- **recv_MAC**(128 bits)
Outline

1 Parameters for the IoT
2 Permutations!
3 Keyed applications
4 STROBE
5 KETJE and KEYAK
6 KRAVATTE and the Farfalle construction
KETJE goals

- Nonce-based AE function
- 96-bit or 128-bit security (incl. multi-target)
- Sessions of header-body pairs
  - keeping the state during the session
- Small footprint
- Target niche: secure channel protocol on secure chips
  - banking card, ID, (U)SIM, secure element, FIDO, etc.
  - secure chip has strictly incrementing counter
- Using reduced-round KECCAK-\(f[400]\) or KECCAK-\(f[200]\), to allow
  - implementation re-use
  - cryptanalysis re-use
  - reasonable side-channel protections
**KETJE instances and lightweight features**

<table>
<thead>
<tr>
<th>feature</th>
<th><strong>KETJE JR</strong></th>
<th><strong>KETJE Sr</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>state size</td>
<td>25 bytes</td>
<td>50 bytes</td>
</tr>
<tr>
<td>block size</td>
<td>2 bytes</td>
<td>4 bytes</td>
</tr>
<tr>
<td><strong>processing</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initialization</td>
<td>12 rounds</td>
<td>12 rounds</td>
</tr>
<tr>
<td>wrapping</td>
<td>1 round</td>
<td>1 round</td>
</tr>
<tr>
<td>8-byte tag comp.</td>
<td>9 rounds</td>
<td>7 rounds</td>
</tr>
<tr>
<td>per session</td>
<td></td>
<td></td>
</tr>
<tr>
<td>per block</td>
<td></td>
<td></td>
</tr>
<tr>
<td>per message</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>computational cost</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

KEYAK goals

- Nonce-based AE function
- 128-bit security (incl. multi-target)
- Session of header-body pairs
  - keeping the state during the session
- Optionally parallelizable
- Conservative safety margin
- Using reduced-round KECCAK-$f[1600]$ or KECCAK-$f[800]$, to allow
  - implementation re-use
  - cryptanalysis re-use
  - reasonable side-channel protections
KEYAK in a nutshell

- SUV = Secret and Unique Value
**KEYAK in a nutshell**

- SUV = Secret and Unique Value
**KEYAK in a nutshell**

- **SUV** = Secret and Unique Value
KEYAK in a nutshell

- SUV = Secret and Unique Value
Leakage robustness

- SUV = Secret and Unique Value
- Provided that **uniqueness** is enforced
- then ...
  - the secret state is a *moving target* [Taha, Schaumont, HOST 2014]
Outline

1. Parameters for the IoT
2. Permutations!
3. Keyed applications
4. STROBE
5. KETJE and KEYAK
6. KRAVATTE and the Farfalle construction
KRAVATTE and the Farfalle construction

The new Farfalle construction

\[ K \parallel 10^* \rightarrow p_b \]

\[ m_0 \rightarrow p_c \]

\[ m_1 \rightarrow p_c \]

\[ m_i \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ K \parallel 10^* \rightarrow p_b \]

\[ m_0 \rightarrow p_c \]

\[ m_1 \rightarrow p_c \]

\[ m_i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ k' \rightarrow p_e \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ i \rightarrow p_c \]

\[ k \rightarrow p_c \]

\[ k \rightarrow p_c \]
KRAVATTE for many purposes

KRAVATTE = Farfalle + Keccak-$p[1600]$

<table>
<thead>
<tr>
<th>KRAVATTE-PRF</th>
<th>Authentication</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRAVATTE-SAE</td>
<td>Session authenticated encryption</td>
</tr>
<tr>
<td>KRAVATTE-SIV</td>
<td>Synthetic-IV authenticated encryption</td>
</tr>
<tr>
<td>KRAVATTE-WBC</td>
<td>Wide block cipher, authenticated encryption with minimal expansion</td>
</tr>
</tbody>
</table>
Conclusions

- **Permutations** are well suited for IoT devices, especially for
  - code size
  - memory usage
- **Farfalle** brings efficiency also on the high-end server
- Bear in mind protections against **side-channel attacks**
Thanks for your attention!

Any questions?

https://keccak.team/

@KeccakTeam
A very classical example

RSA:

\[ c^d \mod n = m \]

Implemented using the square \& multiply algorithm:

http://www.embedded.com/print/4199399
How to protect against side-channel attacks?

- **Electrical-level countermeasures**
  - E.g., balancing the processing of 0 and 1

- **System-level countermeasures**
  - E.g., limit the use of a key

- **Algorithmic countermeasures**
  - Randomization
  - E.g., instead of processing $x$, process $y$ and $z$ s.t. $x = y \oplus z$
What block cipher are used for?

- Hashing: Davies-Meyer, ...
- Block encryption: ECB, CBC, ...
- Stream encryption:
  - synchronous: counter mode, OFB, ...
  - self-synchronizing: CFB
- MAC computation: CBC-MAC, C-MAC, ...
- Authenticated encryption: OCB, GCM, CCM ...
Block cipher operation

Backup slides
Block cipher operation: the inverse
When do you need the inverse?

- Hashing and its modes HMAC, MGF1, ...
- **Block encryption**: ECB, CBC, ...
- **Stream encryption**:
  - synchronous: counter mode, OFB, ...
  - self-synchronizing: CFB
- **MAC computation**: CBC-MAC, C-MAC, ...
- **Authenticated encryption**: OCB, GCM, CCM ...
Block cipher internals
Hashing using Davies-Meyer
Removing diffusion restrictions
Simplifying the view: iterated permutation
Pseudo-random function (PRF)
Message authentication code (MAC)
Stream cipher

nonce
plaintext \oplus \text{plaintext} = \text{ciphertext}
Authenticated encryption

nonce

plaintext = ciphertext

plaintext ⊕ ciphertext = ciphertext

plaintext
Incrementality
Incrementality

packet #1

packet #2

packet #1

packet #2
Incrementality
In-place processing

Store $A[x, y]$ at round $i$ in $(x', y')$ with

$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}^i \begin{pmatrix} x \\ y \end{pmatrix}$.

- Interacts with $\pi$: the output of $\chi$ can overwrite its input
- Matrix of order 4
  - $\Rightarrow$ no performance loss if 4 rounds unrolled

[Bertoni et al., KECCAK implementation overview]
In-place processing

Store $A[x, y]$ at round $i$ in $(x', y')$ with

$$
\begin{pmatrix}
  x' \\
  y'
\end{pmatrix}
= \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}^i
\begin{pmatrix}
  x \\
  y
\end{pmatrix}.
$$

- Interacts with $\pi$: the output of $\chi$ can overwrite its input
- Matrix of order 4
  - $\Rightarrow$ no performance loss if 4 rounds unrolled

[Bertoni et al., KECCAK implementation overview]
In-place processing

Store $A[x, y]$ at round $i$ in $(x', y')$ with

$$
\begin{pmatrix}
  x' \\
  y'
\end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}^i \begin{pmatrix} x \\ y \end{pmatrix}.
$$

- Interacts with $\pi$: the output of $\chi$ can overwrite its input
- Matrix of order 4
  - $\Rightarrow$ no performance loss if 4 rounds unrolled

[Bertoni et al., KECCAK implementation overview]