Permutation Based Cryptography for IoT

Guido Bertoni[1]

Joint work with
Joan Daemen[1], Michaël Peeters[2] and Gilles Van Assche[1]

[1]STMicroelectronics [2]NXP Semiconductors

CIoT 2012, Antwerp, November 21

Motivation

Propose a cipher suite based on a single permutation and a public key primitive for the Internet of Things

Internet of Things Cryptographic Requirements

- One possibility for Internet of Things is the adoption of the Datagram Transport Layer Security
  - Kind of adaptation of TLS for UDP
- Other possibilities, but overall DTLS can be seen as a good example of crypto requirements
- What we report here for DTLS can be easily adapted to other security protocols

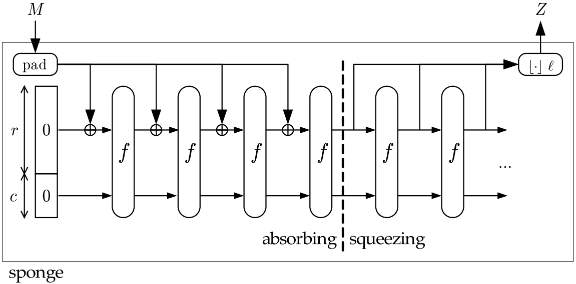
(D)TLS cipher suite

- One of the suggested cipher suite for DTLS and TLS is the ECCGCM [RFC5289]
  - ECC for DH key agreement and digital signature
  - SHA2 for hash and HMAC for PRF
  - AES and GHASH for authenticated encryption

Simplification

- Three different symmetric primitives
  - A luxury that low-end devices would love to avoid!
- Use just one permutation for:
  - hashing
  - authenticated encryption
  - pseudo random number generation
  - key derivation function

Permutation-based construction: sponge

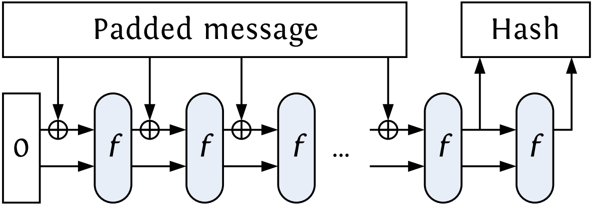


- f: a b-bit permutation with $b = r + c$
    - efficiency: processes r bits per call to f
    - security: provably resists generic attacks up to $2^{c/2}$
- Flexibility in trading rate r for capacity c or vice versa

What can we say about sponge security

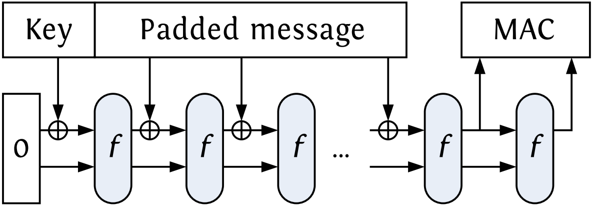
- Generic security:
  - assuming f has been chosen randomly
  - covers security against generic attacks
  - construction as sound as theoretically possible
- Security for a specific choice of f
  - security proof is infeasible
  - Hermetic Sponge Strategy
  - design with attacks in mind
  - security based on absence of attacks despite public scrutiny

Regular hashing



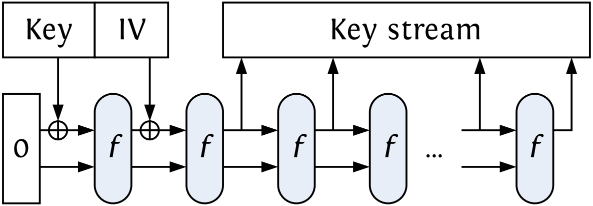
- Pre-sponge permutation-based hash functions
    - Truncated permutation as compression function: Snefru [Merkle '90], FFT-Hash [Schnorr '90], ...MD6 [Rivest et al. 2007]
    - Streaming-mode: SUBTERRANEAN, PANAMA, RADIOGATÚN, Grindahl [Knudsen, Rechberger, Thomsen, 2007], ...

Message authentication codes



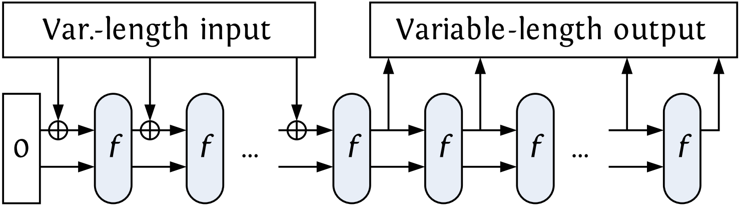
- Pre-sponge (partially) permutation-based MAC function: Pelican-MAC [Daemen, Rijmen 2005]

Stream encryption

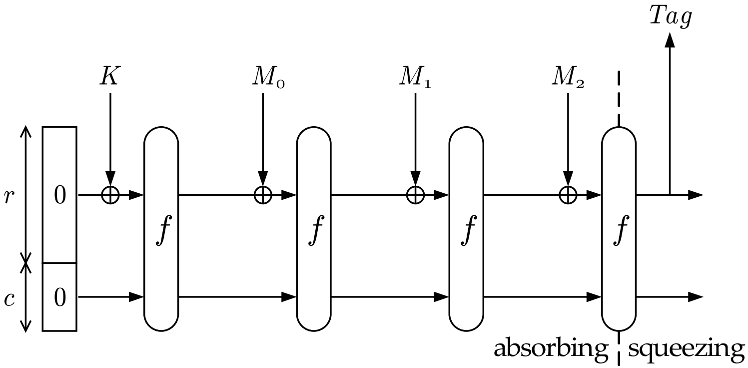


- Similar to block cipher modes:
  - Long keystream per IV: like OFB
  - Short keystream per IV: like counter mode
- Independent permutation-based stream ciphers: Salsa and ChaCha [Bernstein 2007]

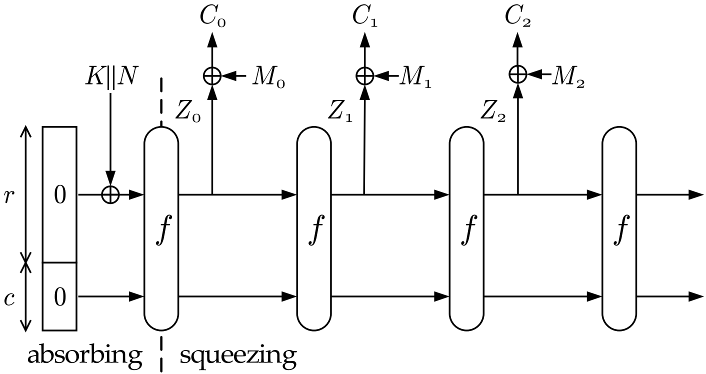
Mask generating function



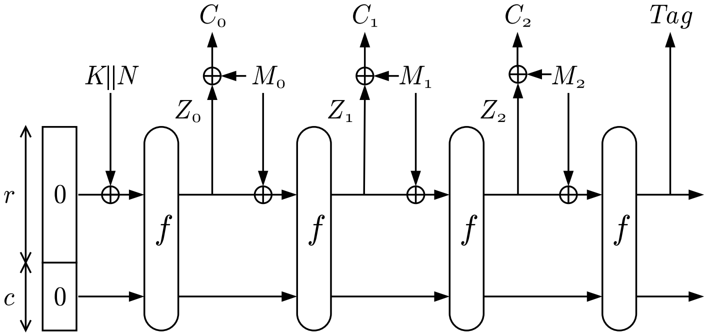
Authenticated encryption: MAC generation



Authenticated encryption: encryption

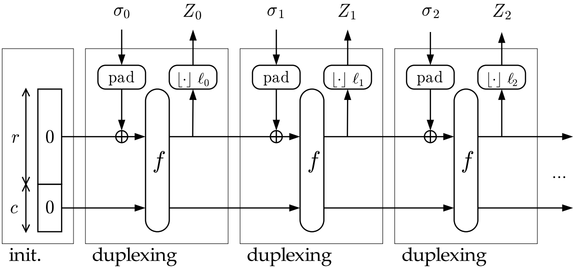


Authenticated encryption: just do them both?



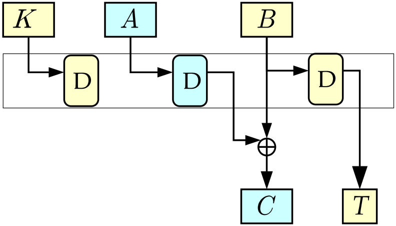
Absorbing and squeezing

The duplex construction



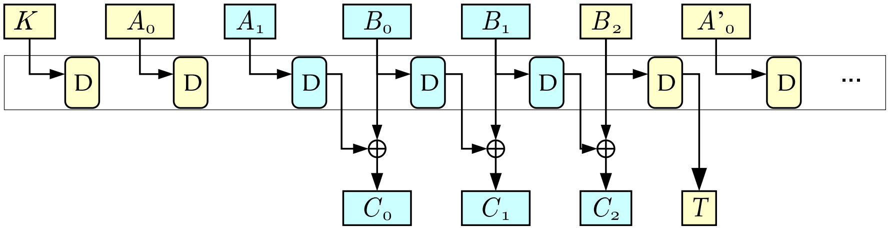
- Object: $D = \textsc{duplex}[f, \text{pad}, r]$
- Requesting ℓ-bit output $Z = D.\text{duplexing}(\sigma, \ell)$
- Generic security equivalent to that of sponge

SpongeWrap authenticated encryption



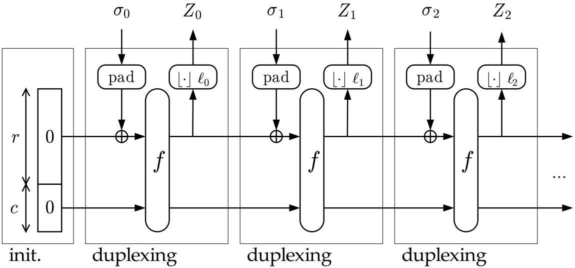
- Single-pass authenticated encryption
- Processes up to r bits per call to f
- Functionally similar to (P)helix [Lucks, Muller, Schneier, Whiting, 2004]

The SpongeWrap mode



- Key *K*, data header *A* and data body *B* of arbitrary length
- Confidentiality assumes unicity of data header
- Supports intermediate tags

The SpongeWrap mode



- SpongeWrap, two simple operations:
  - *D*.initialize()
  - *D*.duplexing(σ, ℓ)
- Frame bits for separating the different stages [SAC 2011]

Sponge functions exists!

KECCAK	Bertoni, Daemen, Peeters, Van Assche	SHA-3 2008	25, 50, 100, 200 400, 800, 1600
Quark	Aumasson, Henzen, Meier, Naya-Plasencia	CHES 2010	136, 176 256, 384
Photon	Guo, Peyrin, Poschmann	Crypto 2011	100, 144, 196, 256, 288
Spongent	Bogdanov, Knezevic, Leander, Toz, Varici, Verbauwhede	CHES 2011	88, 136, 176 248, 320

The lightweight taste

- Quark, Photon, Spongent: *lightweight hash functions*
- Lightweight is synonymous with low-area
- Easy to see why. Let us target security strength $2^{c/2}$
  - Davies-Meyer block cipher based hash ("narrow pipe")
    - chaining value (block size): $n \geq c$
    - input block size (*key* length): typically $k \geq n$
    - feedforward (block size): n
    - total state $\geq 3c$
  - Sponge ("huge state")
    - permutation width: $c + r$
    - r can be made arbitrarily small, e.g. 1 byte
    - total state $\geq c + 8$

Permutations vs block ciphers

- Unique block cipher features
  - pre-computation of key schedule
    - storing expanded key costs memory
    - may be prohibitive in resource-constrained devices
  - misuse resistance
    - issue: keystream re-use in stream encryption
    - not required if nonces are affordable or available
- Unique permutation features
  - diffusion across full state
  - flexibility in choice of rate/capacity

Boosting keyed permutation modes

- Taking a closer look at rate/capacity trade-off
  - keyed generic security is $c - a$ instead of $c/2$
  - with 2^a ranging from data complexity down to 1
  - allows increasing the rate
- Distinguishing vulnerability in keyed vs unkeyed modes
  - in keyed modes attacker has less power
  - allows decreasing number of rounds in permutation

Numeric example

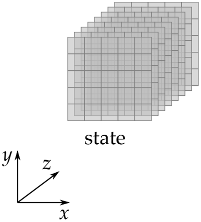
- Say we have the following requirements:
  - we have a permutation with width 200 bits
  - we want to realize different functions
  - desired security strength: 80 bits
  - we assume active adversary, limited to 2^{48} data complexity
- Collision-resistant hashing: $c = 2 \times 80 \Rightarrow r = 40$
- SpongeWrap: $c = 80 + 48 + 1 \Rightarrow r = 71$
- MAC computation: $c = 80 \Rightarrow r = 120$

Unkeyed modes weaker than keyed modes?

- MD5 hash function [Rivest 1992]
  - unkeyed: collisions usable in constructing fake certificates [Stevens et al. 2009]
  - keyed: very little progress in 1st pre-image generation
- PANAMA hash and stream cipher [Clapp, Daemen 1998]
  - unkeyed: instantaneous collisions [Daemen, Van Assche 2007]
  - keyed: stream cipher unbroken till this day
- KECCAK crypto contest with reduced-round challenges
  - unkeyed: collision challenges up to 4 rounds broken [Dinur, Dunkelman, Shamir 2012]
  - keyed: 1st pre-image challenges up to 2 rounds broken [Morawiecki 2011]

KECCAK-f: the permutations in KECCAK

Operates on 3D state:



- (5×5)-bit slices
- 2^{ℓ}-bit lanes
- param. $0 \leq \ell < 7$

- Round function with 5 steps:
  - θ: mixing layer
  - ρ: inter-slice bit transposition
  - π: intra-slice bit transposition
  - χ: non-linear layer
  - ι: round constants
- Lightweight, but high diffusion
- $\#$ rounds: $12 + 2\ell$ for $b = 2^{\ell}25$
  - 12 rounds in KECCAK-$f[25]$
  - 24 rounds in KECCAK-$f[1600]$
- High safety margin, even if unkeyed

KECCAK: reference versions

- KECCAK with default parameters: KECCAK[]
  - width $b = 1600$: largest version
  - rate $r = 1024$: power of 2
  - gives generic security strength $c/2 = 288$ bits
  - roughly 7 % slower than the KECCAK SHA-3 256-bit candidate
  - For performance see eBash, Athena, XBX, etc.
- KECCAK[r=40, c=160]
  - width $b = 200$: small state
  - $c = 160$, generic security strength 80 bits
  - gives rate of $r = 40$
  - roughly 2.4 more work per input/output bit than KECCAK[]

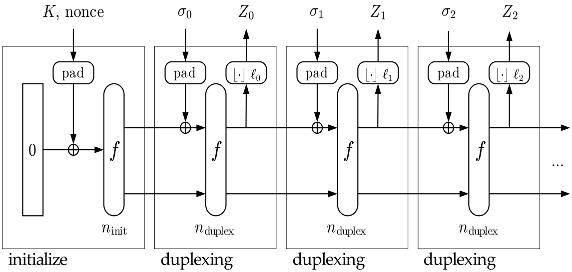
Reduced-round versions of KECCAK: KECCUP

- For keyed modes use reduced-round versions of KECCAK-f
  - called KECCUP$[r, c, n]$ and KECCUP-$f[b, n]$
  - we assume that the multiplicity 2^a is below 2^{64}
- KECCUP for IoT
  - state $b = 200$
  - rate $r = 16$
  - \# rounds ... see next slides

Introducing dedicated variants

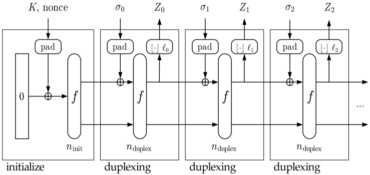
- Sponge and duplex are generic constructions
    - flexible and multi-purpose
    - do not exploit mode-specific adversary limitations
- MAC computation
    - before squeezing adversary has no information about state
    - relaxes requirements on f during absorbing
- Authenticated encryption in presence of nonces
    - nonce can be used to *decorrelate* computations
- Presented at [DIAC2012]

The monkeyDuplex construction



- For authenticated encryption and keystream generation
- Initialization: key, nonce and strong permutation
- reduced number of rounds in duplex calls

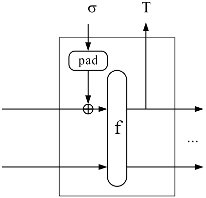
Some monkeyDuplex KECCAP varieties



- $n_{\text{init}} = 12$: dictated by chosen-input-difference attacks
- For $b = 200$ we proposed $n_{\text{duplex}} = 1$: streaming mode

b	$\lvert K\rvert$	c	r	n_{duplex}	n_{init}	speedup
200	80	184	16	1	12	7.2

Consideration 1: monkeyDuplex and MAC generation



- Reduced number of round could give a low propagation from last input block to first squeezed block
    - Attack: change one (or few) bits in the last block of the ciphert text and adapt the MAC with high probability
    - Considered for donkeySponge (MAC) overlooked for monkeyDuplex (AE)

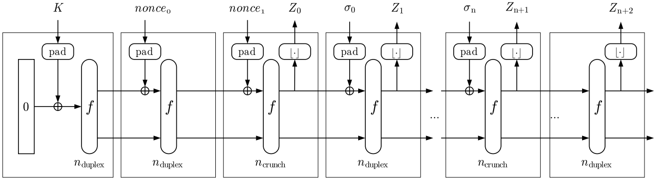
Consideration 1: monkeyDuplex and MAC generation

- The propagation of the duplex should be careful analysed
- Add a sufficient number of rounds before squeezing MAC
  - Gives good diffusion and reduces the possibilities of the attacker
- If the size of the MAC is larger than the rate, the nominal duplex round is applied after the first block of MAC

Consideration 2: monkeyDuplex and Key + Nonce size

- In the original proposal the size of (key + nonce) $< b$
- Depending on the size of b and protocol this might be too restrictive
- Review of the initial phase of the scheme as well

Reviewing monkeyDuplex work in progress



- Define three interfaces of the duplex object
  - *D.initialize*(K)
  - *D.crunching*(σ, ℓ): used to separate different phases
  - *D.duplexing*(σ, ℓ): all other cases
- The difference is the number of rounds of the KECCUP-*f*

Practical proposals

- Public key, like ECC P192 (why this? see next line..)
- KECCAK[r=8, c=192] as hash function for digital signature
- KECCAK[r=8, c=192] for PRF
    - rate can be increased to 40 bits if needed
- monkeyDuplex
    - $D.initialize(K)$: KECCUP[r=16, c=200, n=1]
    - $D.crunching(\sigma, \ell)$: KECCUP[r=16, c=200, n=6]
    - $D.duplexing(\sigma, \ell)$: KECCUP[r=16, c=200, n=1]

Performances

- Two interesting papers will be presented at Cardis 2012:
    - Yalcin et al ”On the Implementation Aspects of Sponge-based Authenticated Encryption for Pervasive Devices”
    - Balasch et al ”Compact Implementation and Performance Evaluation of Hash Functions in ATtiny Devices” (presented yesterday by Tim)

Performances comparison in Software

- What do you gain on ATtiny?

Algorithm	RAM	code size	cycle (10^3) (500 byte message)
KECCAK[]	244	868	716
KECCAK[r=40, c=160]	48	752	1206
this proposal	48	752	180
AES v1	33	1659	140
AES Furios	192	1568	113

AES performances extrapolated from ECRYPT II web page (include multiple key schedules but no data integrity)

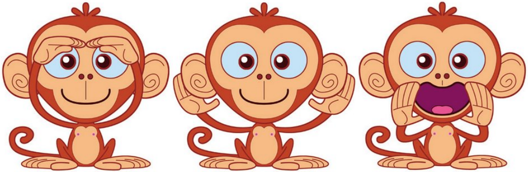
Performances comparison in Hardware

- What do you gain in hardware?

Algorithm	kGate	cycle per byte
KECCAK[]	10	5
KECCAK[r=40, c=160]	6.5	3.6
this proposal	6.5	0.5
AES	2.4	8.6

[Keccak Implementation] 130nm, area can be reduced increasing computational time
For AES only encryption no data integrity

Don’t forget, the Sponge can forget



If you are worried about ”midgame” [crypto 2012 rump session] where a powerful attacker can read your entire intermediate state but not your keys you may want to use the forget or overwrite mode.

Conclusions and Future Work

- Single permutation and a public key primitive satisfy all the cryptographic requirements of IoT
- Performance point of view: the monkeyDuplex seems very attractive primitive
  - detailed analysis of the number of round per permutation is highly recomended
- 400 bit permutation for 128 bit security against collision resistance?
- public key based on Sponge, we wish...

Questions?

Thanks for your attention!

Q?

More information on
http://keccak.noekeon.org/
http://sponge.noekeon.org/