Developments in Authenticated Encryption

Dr. Markku-Juhani O. Saarinen

mjos@item.ntnu.no

NTNU

Norwegian University of Science and Technology

Finse Winter School, 05 May 2014

1. Layman's Introduction to Authenticated Encryption

Authenticated Encryption

An Authenticated Encryption (AE) primitive provides **both**:

- **Authentication** or message integrity protection, and
- **Encryption** or confidentiality / privacy protection.

Preferably in a single processing step, allowing parallelism.

Security protocols such as IPSec and SSL/TLS usually required two processing steps for each packet in 1990's and 200x's.

- **Authentication** was handled with HMAC (Hash Message Authentication Code), which builds a MAC from a hash H:

$$\text{SQ HMAC}(K, m) = H(K \oplus \texttt{opad} \mid H(K \oplus \texttt{ipad} \mid m)).$$

- **Encryption** was provided either with block cipher such as 3DES-CBC or AES-CBC or a stream cipher such as RC4.

Hardware implementation of such a twin set-up is cumbersome.

This is still the standard mechanism in TLS 1.3, although transition to authenticated encryption has been swift during last few years.

Encrypt-then-MAC vs. MAC-then-Encrypt (1)

M. Bellare and C. Namprempre: "Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm" ASIACRYPT 2000.

This paper was quite influential about 10+ years ago, but the results have been overstated and often misinterpreted.

It claimed that Encrypt-and-MAC (i.e. any modern Authenticated Encryption algorithm) "does not preserve privacy"

> *.. because the MAC could reveal information about the plaintext.*

Solution: Just use any (randomized) MAC that does not reveal such information.

Encrypt-then-MAC vs. MAC-then-Encrypt (2)

M. Bellare and C. Namprempre**y**:

> *E & M also fails to provide integrity of ciphertexts. This is because there are secure encryption schemes with the property that a ciphertext can be modified without changing its decryption.*

Solution: Show that your encryption scheme does not have that property. For example all stream ciphers are bijective and do not have that property; Integrity of ciphertext exists for those.

One can reasonably argue that the models (for "secure encryption schemes" and "secure MACs") that Bellare & Co. *chose to use* were broken, not Encrypt-and-MAC itself.

Authenticated Encryption with Associated Data

An Authenticated Encryption with Associated Data (AEAD) primitive additionally allows associated (plaintext) data to be authenticated (integrity protection only).

Inputs:

- **Secret Key** for both confidentiality and integrity.
- **Initialization Vector** (IV) which is typically a *nonce*.
- **Associated Data** which is not encrypted, only authenticated.
- **Plaintext** which is encrypted for confidentiality protection, and also authenticated.

Outputs:

- **Ciphertext**, which corresponds to the plaintext above.
- **Authentication Code**, against modification of Ciphertext or Associated Data.

(Ciphertext and MAC may be intermixed.)

Role of Associated Data (1)

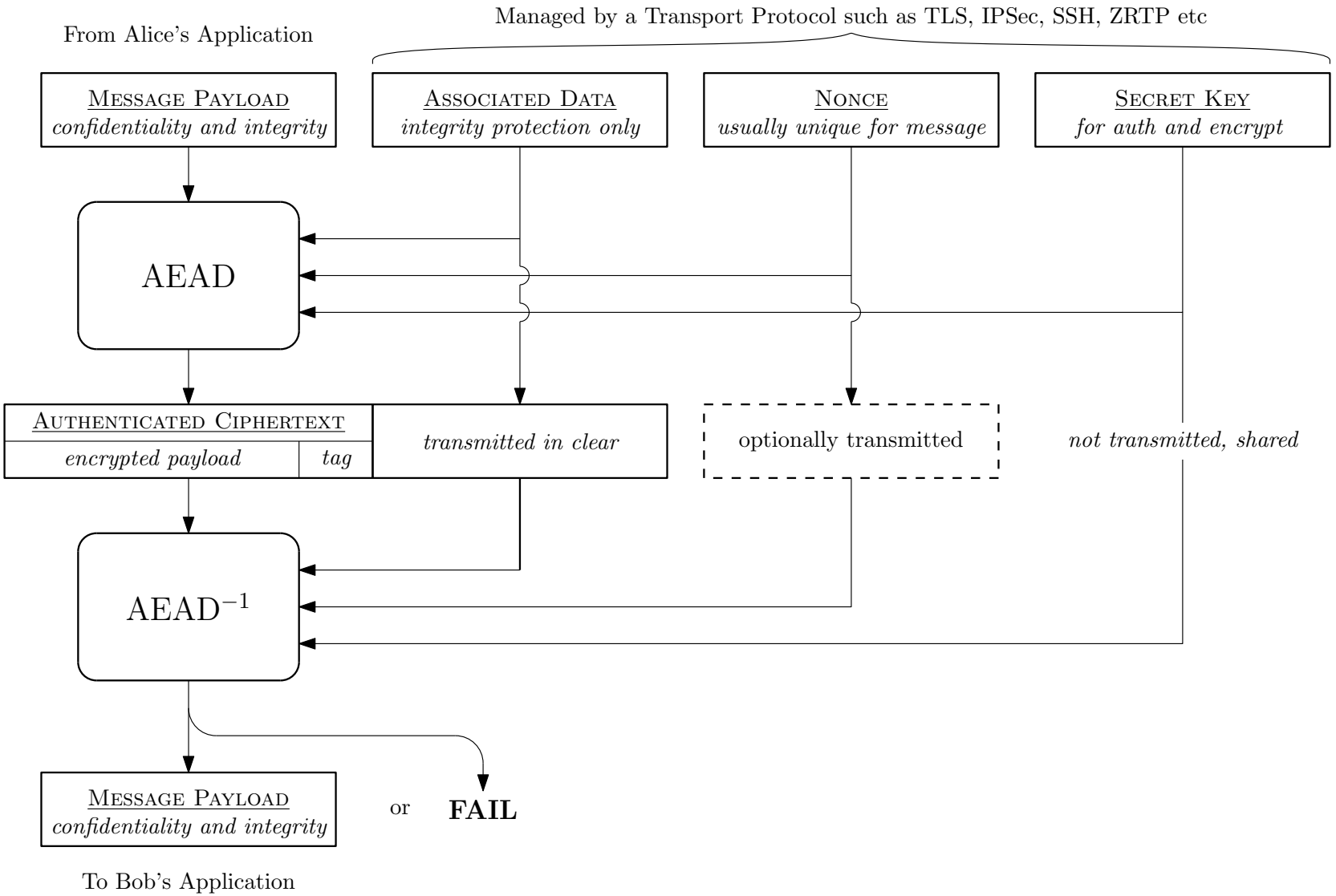
Authenticated Associated Data (AAD) typically consists of information such as:

- Session identifier or Security Association
- Source and destination address of message
- Sequence number and other information stream reassembly

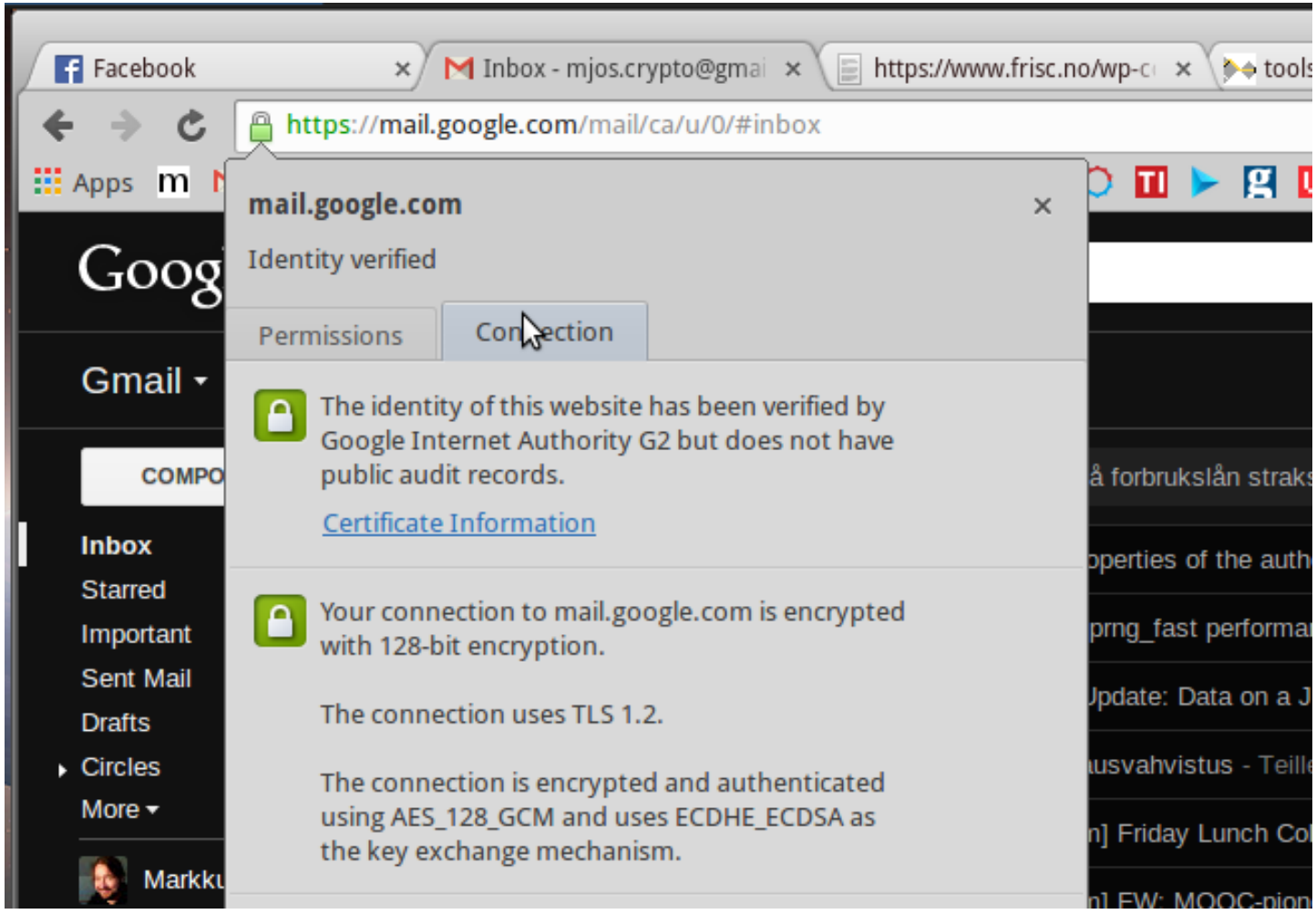
This information must be transmitted in clear for correct delivery, but its content must still be integrity protected with an authentication key for security.

Associated Data is typically managed by the secure transport protocol or layer, and is transmitted in clear.

Role of Associated Data (2)



2010s: With AES-GCM AE is now mainstream



Your gmail, facebook, and VPN probably uses AES-GCM.
GCM: Galois / Counter Mode, FIPS SP 800-38D (2007) is an AEAD mode for AES.

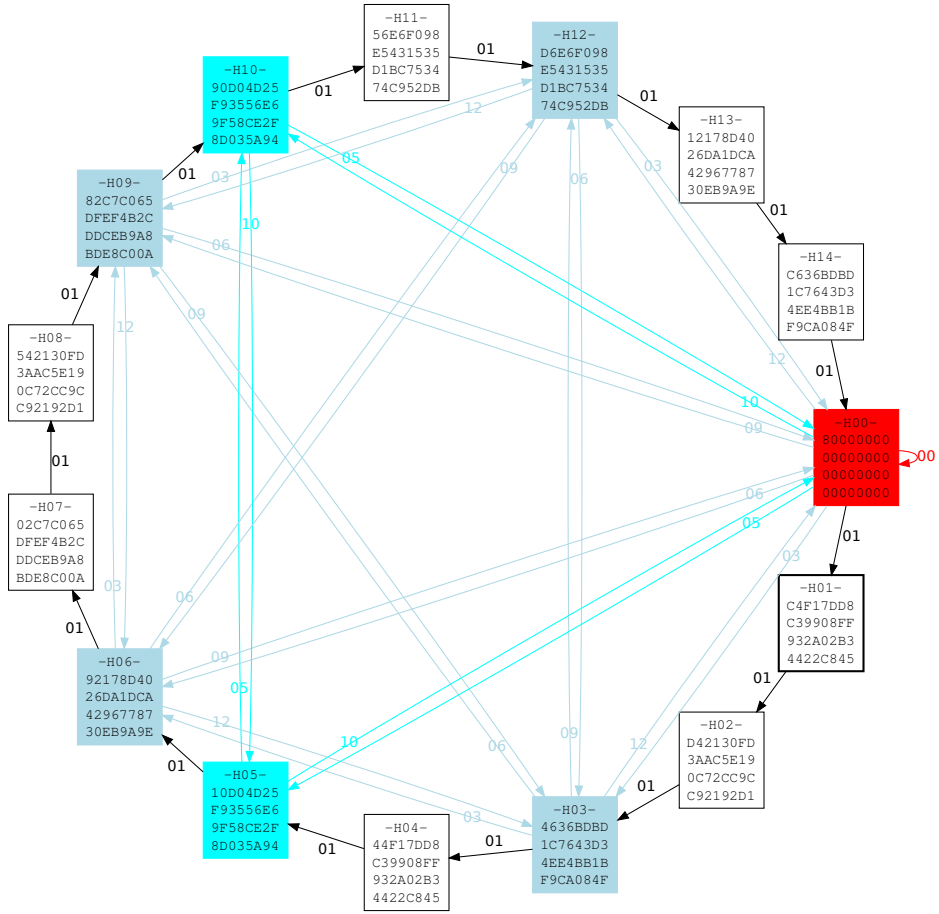
AES-GCM Only Approved "Suite B" Algorithm

AES-GCM is currently the only approved algorithm in NSA's unclassified "Suite B", making it basically the only choice for Governmental and NATO COTS information processing.

Currently certified up to TOP SECRET with 256-bit keys.

Historical Timeline

- Authenticated encryption schemes go back some 30+ years (ANSI X9.9 CBC-MAC, 1981).
- First "provably secure" AEAD mode in year 2000 (IACBC).
- The 2004 total break of MD5 by Xiaoyun Wang & Co. resulted in near-panic with regard to SHA security, which in turn led to the SHA-3 competition (2007-2012).
- Apparently fear for HMAC-SHA security caused NSA to adopt AES-GCM in Suite B (2005) and to issue IETF RFCs for AES-GCM use in SSH, TLS, and IPSec.
- This is of course really quite silly as we know that even HMAC-MD5 is more secure than AES-GCM (later).
- Some fear the the push by NSA to put AES-GCM everywhere is driven by some nefarious motives (I don't think so).
- AES-GCM was applauded by the industry (e.g. CISCO) since AES-GCM is significantly easier/cheaper to implement for high-speed links in hardware. Intel adopted hardware assistance in its processors in the form of AES NIS.



$1\frac{1}{2}$ "Cycling Attacks on GCM, GHASH and Other Polynomial MACs and Hashes"

FSE 2012, Washington D.C. 20 March 2012

2. 21st Century crypto: Sponges, KECCAK, and CAESAR

SHA-3 Algorithm KECCAK: A Sponge-Based Hash

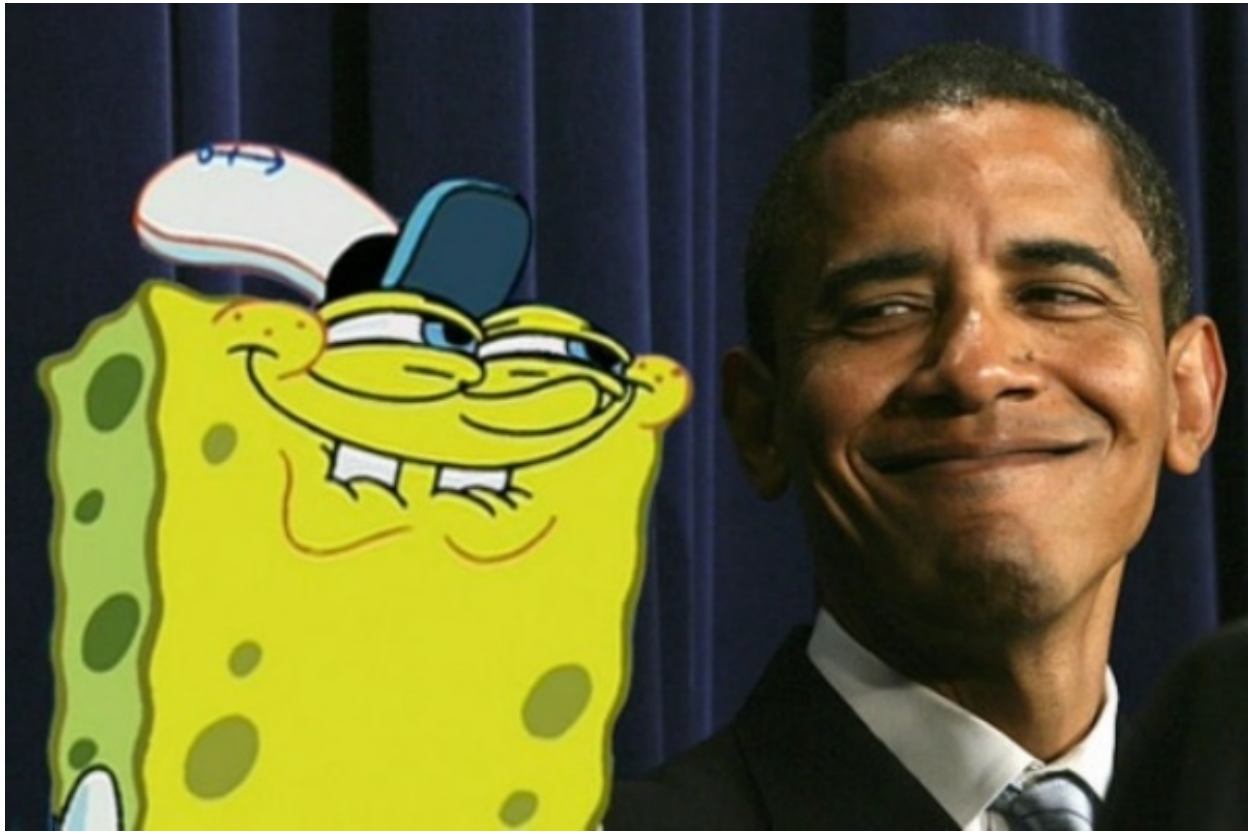
KECCAK and "Sponge Functions" were invented by:

Guido Bertoni (STMicro)
Joan Daemen (STMicro)
Michaël Peeters (NXP)
Giles Van Assche (STMicro)

- **11 Feb 2007** NIST Call for Proposals
- **31 Oct 2008** SHA3 Submissions (64)
- **10 Dec 2008** First Round (54)
- **24 Jul 2009** Second Round (14)
- **09 Dec 2010** SHA3 Finalists (5)
- **12 Aug 2011** KECCAK Team in SAC '11: "Duplexing the sponge.."
- **02 Oct 2012** KECCAK Chosen
- **04 Apr 2014** NIST publishes "DRAFT FIPS 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions"

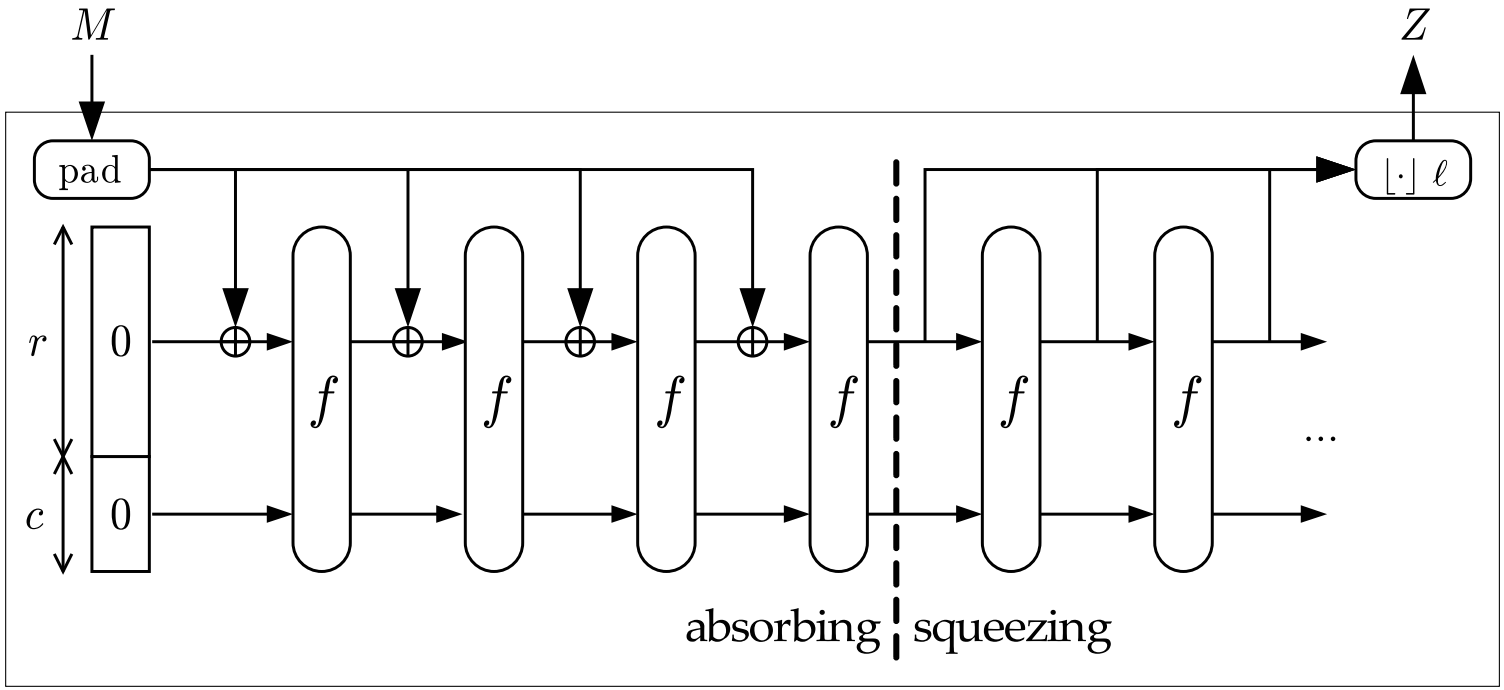
≈ **2015** KECCAK will be officially **S**ecure **H**ash **A**lgorithm 3.

SHA-3 Algorithm KECCAK: An U.S. Standard



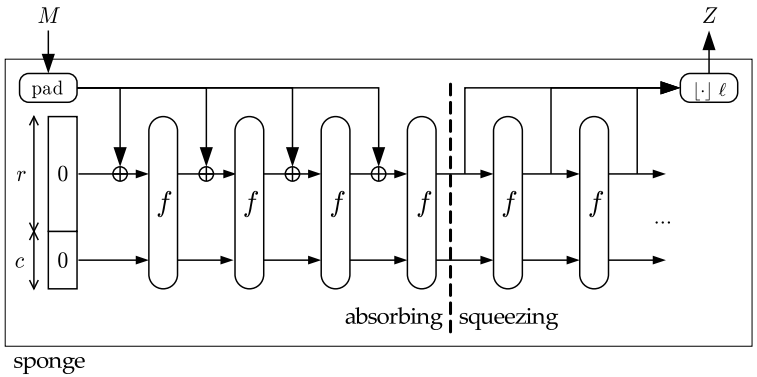
- So.. an European Sponge will become the new U.S. Civilian, Federal Government and Military cryptographic hash standard for digital signatures and other uses.
- And it might also become the standard for Authenticated Encryption !

The Sponge Construction for Hashing



- More general than traditional hash function: arbitrary-length output
- Built from a b-bit permutation f (π commonly used), with $b = r + c$
  - r bits of rate, related to hashing speed
  - c bits of capacity, related to security

Generic Security as a Hash (from Keccak team)



- N is the number of f invocations
- r is the rate in bits
- c is the capacity in bits

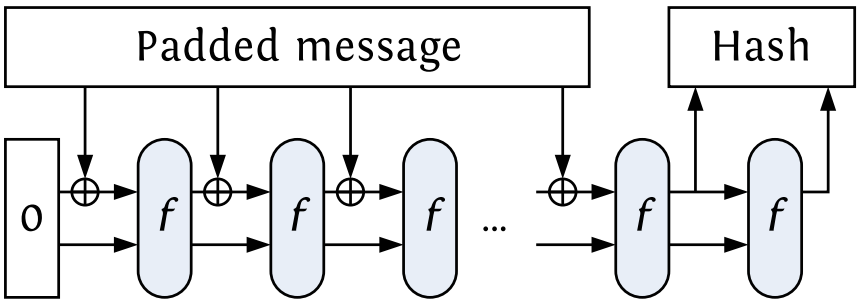
RO-differentiating adv. $\leq N^2/2^{c+1}$

- Proven by Keccak team, Eurocrypt 2008
- As strong as random oracle against attacks with $N < 2^{c/2}$

Bound assumes f to be a **random** permutation

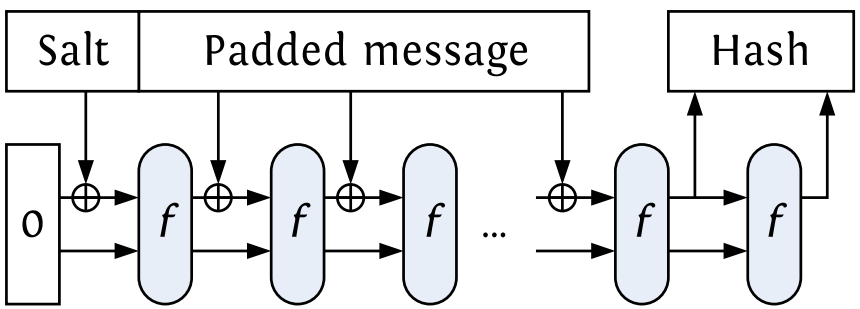
- Covers generic attacks against the mode
- .. but not attacks that exploit specific properties of f

Regular Hashing and Salted Hashing



Unkeyed Hash Mode (SHA-3)

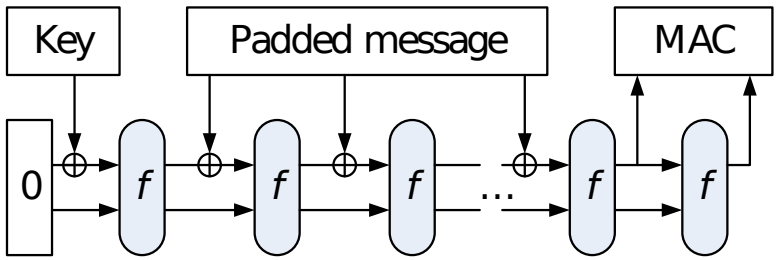
- Electronic signatures (PKCS #1 RSA, ECDSA)
- Data integrity (sha1sum etc)
- Data id (git, anti-virus, P2P)

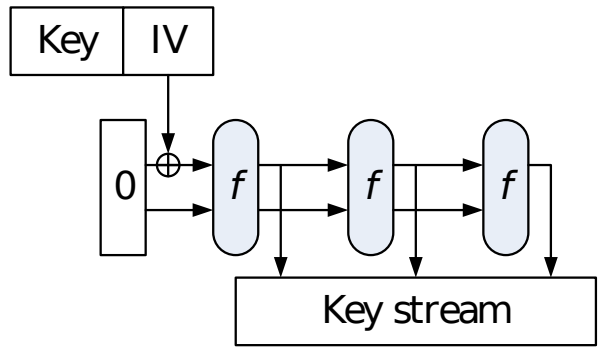


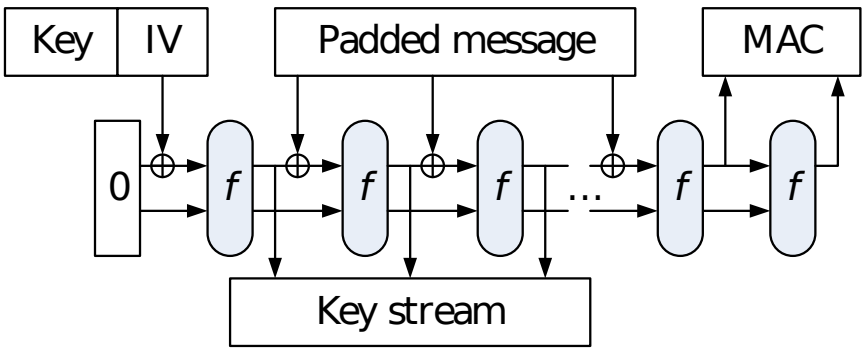
Salted Mode

- Randomized hashing (RSASSA-PSS, Salt = IV)
- Password storage (/etc/shadow)
- Message Authentication Codes (Salt = KEY)
- Different security parameters

Other Provable Uses for Sponge Permutations







MAC Mode: Provably secure.

Stream Cipher Mode: Provably secure.

Authenticated Encryption is encryption + MAC. We'll come back to this later.

DuplexWrap (basic Sponge AE Scheme) Bounds

Theorem

The DuplexWrap *and BLNK authenticated encryption modes satisfy the following privacy and authentication security bounds:*

$$\mathrm{Adv}^{\mathsf{priv}}_{\mathsf{sbob}}(\mathcal{A}) < (M+N)2^{-k} + \frac{M^2+4MN}{2^{c+1}}$$

$$\mathrm{Adv}^{\mathsf{auth}}_{\mathsf{sbob}}(\mathcal{A}) < (M+N)2^{-k} + \frac{M^2+4MN}{2^{c+1}}$$

against any single adversary $\mathcal{A}$ if $K \xleftarrow{\$} \{0,1\}^k$, tags of $l \geq t$ bits are used, and π is a randomly chosen permutation.
M is the data complexity (total number of blocks queried from keyed sponge instance) and N is the time complexity (total number equivalents of π or its inverse execution time).

Proof.

Theorem 4 of [KeyakV1]. See also [AnMePr10,BeDaPeAs11]. □

CAESAR 2014-2017

NIST-Funded **C**ompetition for **A**uthenticated **E**ncryption: **S**ecurity, **A**pplicability, and **R**obustness.
`http://competitions.cr.yp.to/caesar.html`

- **15 Jan 2013** Announced by Dan Bernstein
- **15 Mar 2014** First round submissions (57)
- **15 May 2014** First-round software
- **23-24 Aug 2014** DIAC '14, UCSB
- **15 Jan 2015** Second round candidates
- **15 Feb 2015** Second round tweaks (fixes)
- **15 Dec 2015** Third round candidates
- **15 Dec 2016** Final round candidates
- **15 Dec 2017** Portfolio announcement

CAESAR Situation May 1, 2014

See `https://aezoo.compute.dtu.dk` for updates.

Round 1 Candidates

ACORN *++AE* *AEGIS* *AES-CMCC* ~~*AES-COBRA*~~ *AES-COPA* *AES-CPFB* *AES-JAMBU* *AES-OTR* *AEZ* Artemia Ascon *AVALANCHE* Calico *CBA* ~~CBEAM~~ *CLOC* Deoxys *ELmD* *Enchilada* ~~FASER~~ ~~HKC~~ HS1-SIV ICEPOLE *iFeed[AES]* *Joltik* *Julius* **Ketje** **Keyak** *KIASU* *LAC* *Marble* McMambo Minalpher MORUS NORX *OCB* OMD PAEQ *PAES* PANDA π-Cipher *POET* POLAWIS PRIMATEs Prøst Raviyoyla Sablier *SCREAM* *SHELL* *SILC* *Silver* STRIBOB Tiaoxin TriviA-ck Wheesht *YAES*

- *Based on AES or other Block Cipher*
- **Based on KECCAK** (submissions from KECCAK team)
- ~~Withdrawn from Competition by Authors~~
- Pretty much broken by someone
- Less fatal vulnerabilities reported

On CAESAR Cryptanalysis

I had two "one-man band" candidates:

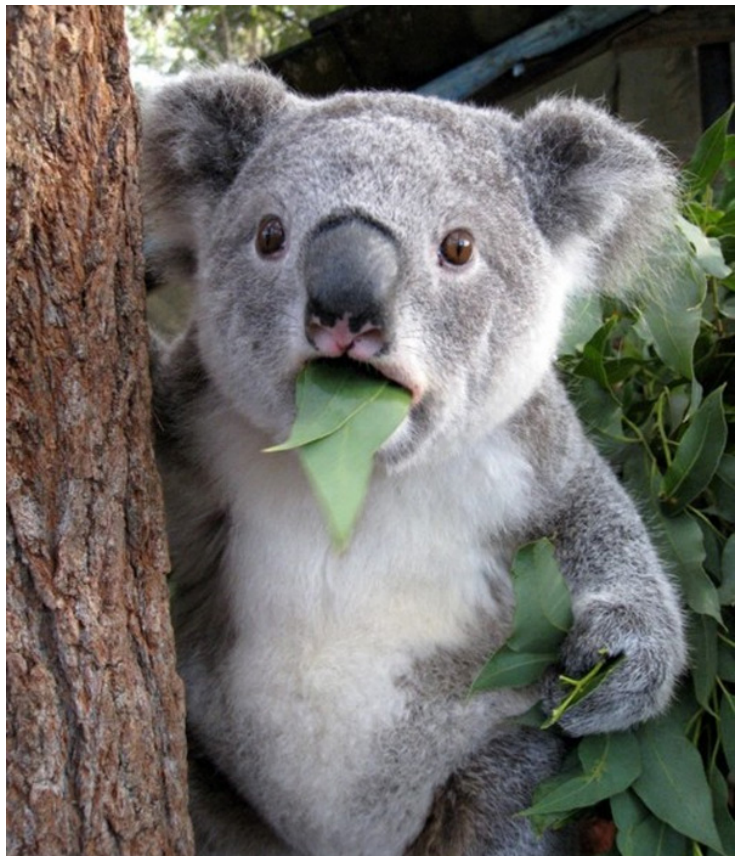
- **CBEAM**, a super-lightweight Sponge based on BLNK and novel ideas like rotation-invariant ϕ functions.
- **STRIBOB**, a high-security Sponge based on BLNK an the LPS transform of GOST R 34.11-2012 LPS "Streebog" Permutation.

On April 3, 2014, Brice Minaud of ANSSI reported a relatively simple differential attack on CBEAM that breaks authentication with 2^{-43} probability. I still can't believe that I missed this attack – CBEAM was already published in CT-RSA '14 (LNCS 8366). ~~CBEAM~~.

On CAESAR Cryptanalysis

I had two "one-man band" candidates:

- **CBEAM**, a super-lightweight Sponge based on BLNK and novel ideas like rotation-invariant ϕ functions.
- **STRIBOB**, a high-security Sponge based on BLNK an the LPS transform of GOST R 34.11-2012 LPS "Streebog" Permutation.



On April 3, 2014, Brice Minaud of ANSSI reported a relatively simple differential attack on CBEAM that breaks authentication with 2^{-43} probability. I still can't believe that I missed this attack – CBEAM was already published in CT-RSA '14 (LNCS 8366). ~~CBEAM~~.

Agence nationale de la sécurité des systèmes d'information, France. They participate in public efforts like CAESAR and inform us about bugs. Don't expect that from NSA GCHQ FSB ..

3. “STRIBOB: Authenticated Encryption from GOST R 34.11-2012 LPS Permutation”

CTCrypt 2014, 05-06 June 2014, Moscow, RUSSIA

GOST R 34.11-2012

Russian Standards (GOST)

- 28149-89 Block Cipher (KGB, 1970s)
- R 34.11-94 was a hash based on 28149-89, used for R 34.10-94 Signatures.
- Cryptanalysis by F Mendel et al (2008): 2^{105} collision, 2^{192} preimage.
- "Streebog" proposed in 2009.
- Since January 1, 2013, the Russian Federation has **mandated** the use of GOST R 34.11-2012 hash algorithm in GOST R 34.10-2012 digital signatures.
- Good security margin.
- **STRIBOB** builds a Sponge AEAD from Streebog!

GOST R 34.11-2012 "Streebog"

The 2012 standard has superficial similarities to the 1994 standard but also features clearly AES-inspired design elements. Intended for Digital Signatures (R 34.10-2012), but also used with HMAC.

A (non-keyed) cryptographic hash function that produces a 256-bit or 512-bit message digest for a bit string of arbitrary length.

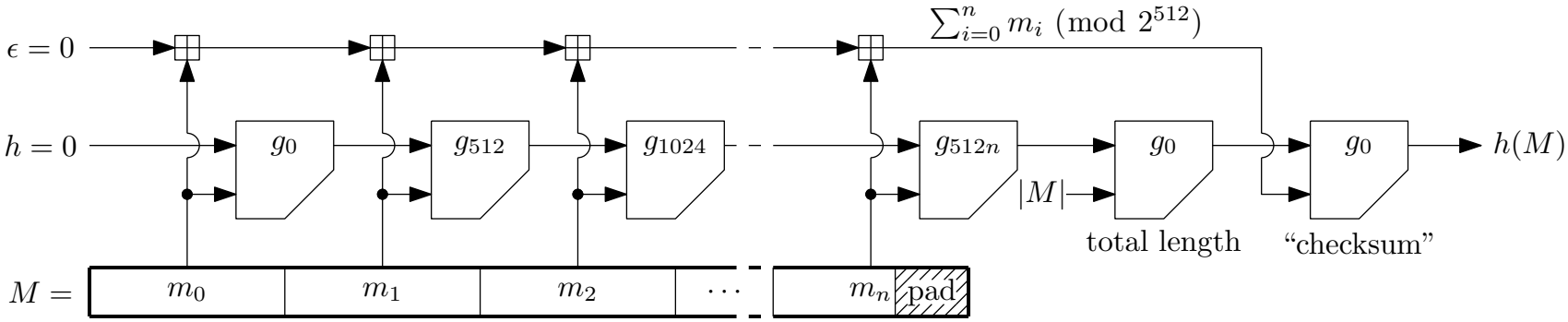
Standard security requirements:

- **Collision resistance**:
  Finding m_1 and m_2, $h(m_1) = h(m_2)$ should require $2^{\frac{n}{2}}$ effort.
- **Pre-image resistance**:
  Finding m for given h in $h = H(m)$ should require 2^n effort.
- **Second pre-image resistance**:
  Finding m_2 for given m_1 with $h(m_1) = h(m_2)$ should require $\frac{2^n}{|m_2|}$ effort.

Not a Sponge, but a Miyaguchi–Preneel - inspired construction:

$$h_i = E_{g(H_{i-1})}(m_i) \oplus h_{i-1} \oplus m_i.$$

Computing $h(M)$ with Compression Function $g_N(h, m)$



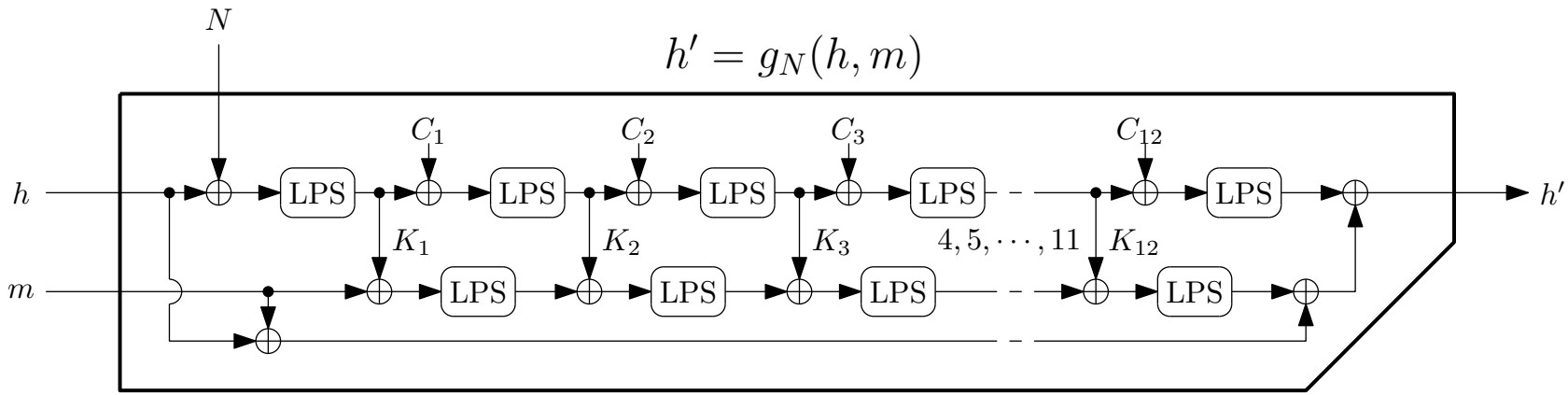
Padded message M is processed in 512-bit blocks $M = m_0 \mid m_1 \mid \cdots \mid m_n$ by a compression function $h' = g_N(h, m_i)$.

For 256-bit hashes, the initial h value is changed to `0x010101..01` and the output $h(M)$ is truncated to 256 bits.

Chaining variable h has 512 bits. N is the bit offset of the block.

There are finalization steps involving two invocations of g, first on the total bit length of M, and then on checksum ϵ, which is computed over all input blocks mod 2^{512}.

Streebog: The Compression Function $g_N(h, m)$



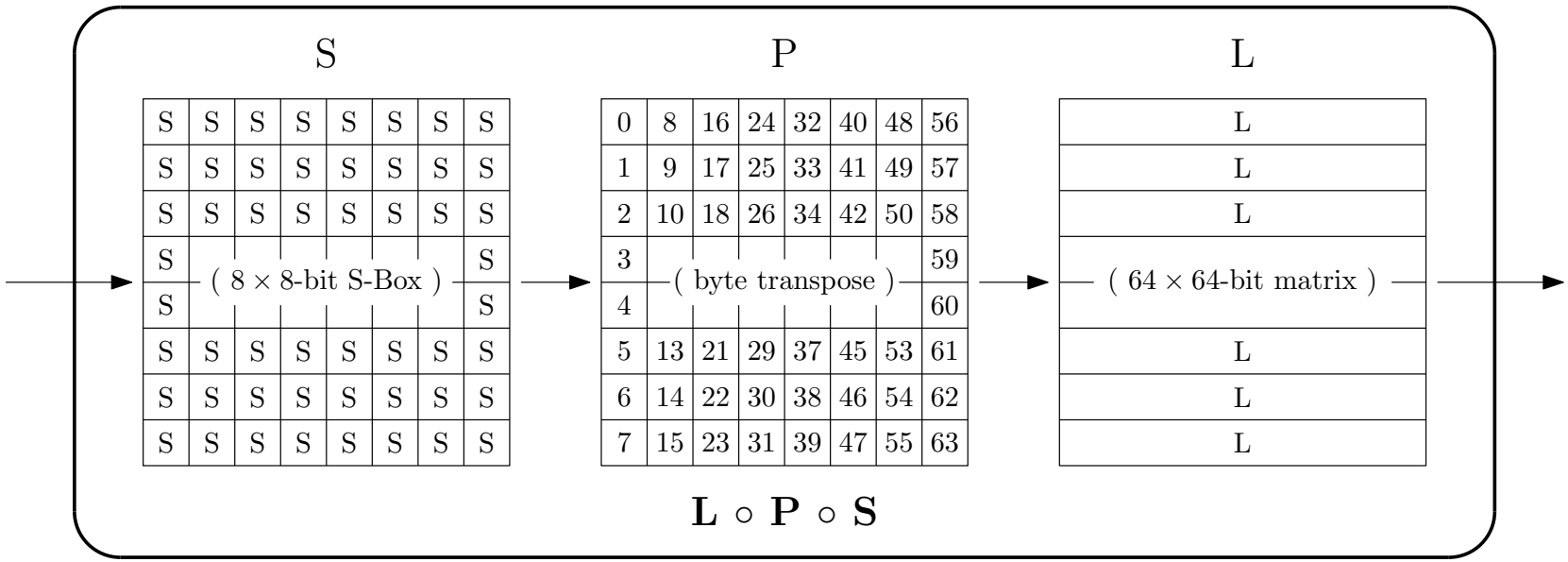
N: bit offset *h*: chaining value *m*: 512-bit message block

The compression function is built form a 512×512 - bit **keyless** permutation LPS and XOR operations. All data paths are 512 bits.

The 12 random round constants C_i are given in the standard spec.

One can see the upper "line" (kinda) keying the lower line via K_i.

Streebog: $\mathrm{LPS} = L \circ P \circ S = L(P(S(x)))$



S : ("Substitution") An 8×8 - bit S-Box applied to each one of 64 bytes ($8 \times 64 = 512$ bits).

P : ("Permutation") Transpose of 8×8 - byte matrix.

L : ("Linear") Mixing of **rows** with a 64×64 binary matrix.

Streebog LPS: 1. Substitution step S

We write the state as zero-indexed matrix $m[0\ldots7][0\ldots7]$, serialized as

$$v[\ i\ ] = m[\ \lfloor i/8 \rfloor\ ][\ i\ \bmod\ 8\ ] \text{ for } i = 0, 1, \cdots, 63.$$

Streebog has a single 8-bit S-box S, as in AES.

In this step, the 8×8 - bit S-Box is applied to each byte (octet):

$$m'[\ i\ ][\ j\ ] = S(m[\ i\ ][\ j\ ]) \ \text{ for } \ 0 \leq i, j \leq 7.$$

S has apparently been designed to resist classical cryptanalysis:

- Differential bound: $P = \frac{8}{256}$
- Best linear approximation: $P = \frac{28}{128}$
- All output bits have Deg 7

Unknown design methodology; apparently not a simple equation like the AES S-Box.

Streebog LPS: 2. Permutation step P

A permutation of bytes in the state, equivalent of transposing the 8×8 byte matrix.

Transposition is a reflection along the main diagonal, which can also be seen as writing rows as columns:

$$m'[\,i\,][\,j\,] = m[\,j\,][\,i\,] \quad \text{for} \quad 0 \leq i, j \leq 7.$$

$$\begin{pmatrix} 00 & 01 & 02 & 03 & 04 & 05 & 06 & 07 \\ 08 & 09 & 0A & 0B & 0C & 0D & 0E & 0F \\ 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 \\ 18 & 19 & 1A & 1B & 1C & 1D & 1E & 1F \\ 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 \\ 28 & 29 & 2A & 2B & 2C & 2D & 2E & 2F \\ 30 & 31 & 32 & 33 & 34 & 35 & 36 & 37 \\ 38 & 39 & 3A & 3B & 3C & 3D & 3E & 3F \end{pmatrix}^{T} = \begin{pmatrix} 00 & 08 & 10 & 18 & 20 & 28 & 30 & 38 \\ 01 & 09 & 11 & 19 & 21 & 29 & 31 & 39 \\ 02 & 0A & 12 & 1A & 22 & 2A & 32 & 3A \\ 03 & 0B & 13 & 1B & 23 & 2B & 33 & 3B \\ 04 & 0C & 14 & 1C & 24 & 2C & 34 & 3C \\ 05 & 0D & 15 & 1D & 25 & 2D & 35 & 3D \\ 06 & 0E & 16 & 1E & 26 & 2E & 36 & 3E \\ 07 & 0F & 17 & 1F & 27 & 2F & 37 & 3F \end{pmatrix}$$

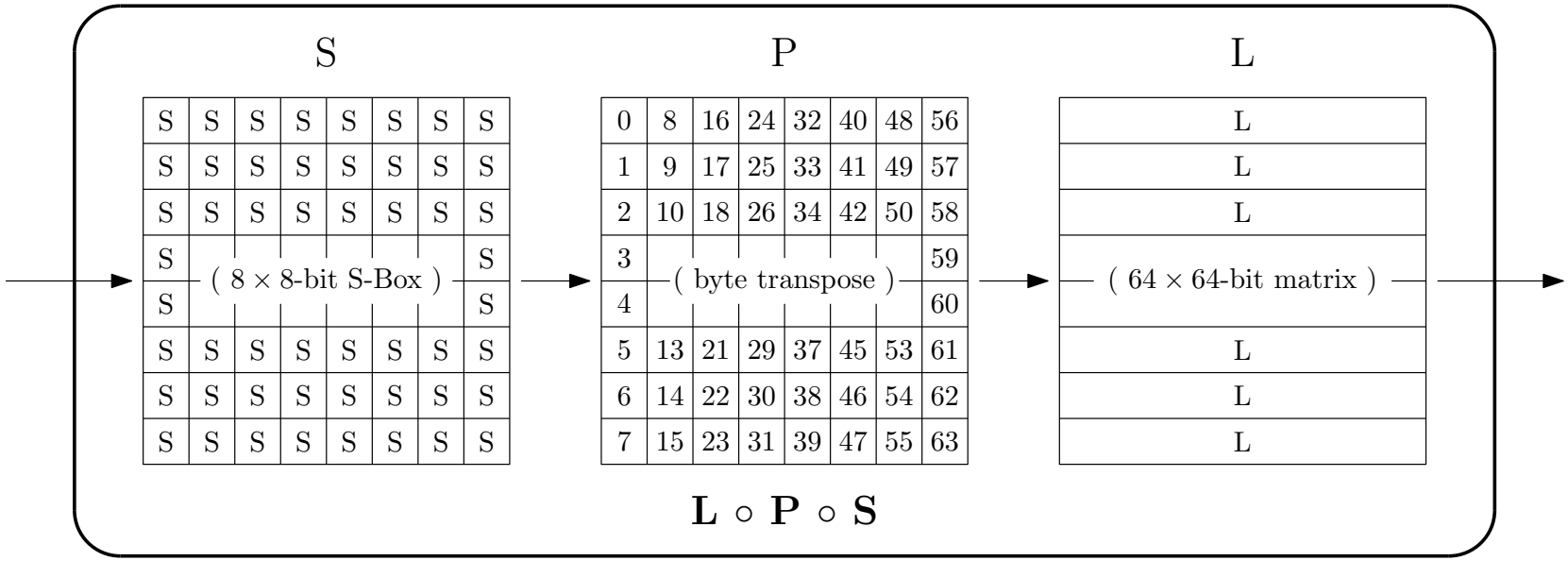
Streebog LPS: 3. Linear step L

L is specified in the form of a 64×64 - bit matrix in the GOST standard text but it is in fact built from a MDS matrix multiplication in the finite field $\mathbb{F}_{2^8}$ with irreducible polynomial $p(x) = x^8 + x^6 + x^5 + x^4 + 1$ in "bit reverse" representation.

$$m' = m \cdot L \quad or \quad m'[\,i\,][\,j\,] = \bigoplus_{k=0}^{7} m[\,i\,][\,k\,] * L[\,k\,][\,j\,],$$

$$L = \begin{pmatrix} \texttt{8E 20 FA A7 2B A0 B4 70} \\ \texttt{A0 11 D3 80 81 8E 8F 40} \\ \texttt{90 DA B5 2A 38 7A E7 6F} \\ \texttt{9D 4D F0 5D 5F 66 14 51} \\ \texttt{86 27 5D F0 9C E8 AA A8} \\ \texttt{45 6C 34 88 7A 38 05 B9} \\ \texttt{E4 FA 20 54 A8 0B 32 9C} \\ \texttt{70 A6 A5 6E 24 40 59 8E} \end{pmatrix}$$

Structural Relationship to AES (“Square design”)



- S ≈ AES `SubBytes`. Larger state: 64 rather than 16 bytes.
- P ≈ AES `ShiftRows`. Each byte will be mixed with every other byte in 2 rounds.
- L ≈ AES `MixColumns`. Rows are mixed with a MDS matrix.

The constant addition corresponds to `AddRoundKey`.

Streebog: Sponge Permutation π

For some vector of twelve 512-bit subkeys C_i we define a 512-bit permutation $\pi_C(X_1) = X_{13}$ with iteration

$$x_{i+1} = \mathsf{LPS}(X_i \oplus C_i) \quad \text{for} \quad 1 \leq i \leq 12.$$

We adopt 12 rounds of LPS as the Sponge permutation with parameters:

- b Permutation size $b = r + c = 512$, the LPS permutation size.
- r Rate $r = 256$ bits.
- c Capacity $c = 256$ bits.

As π satisfies the indistinguishability criteria, we may choose:

- k Key size $k = 192$ bits.
- t Authentication tag (MAC) size $t = 128$ bits.
- k Nonce (IV) size $t = 128$ bits.

Easy Security Reduction

Theorem

If $\pi_C(x)$ can be effectively distinguished from a random permutation for an arbitrary set of constants C_i, so can $g_N(h, x)$ for any h and N.

Proof.

If h is known, so are all of the subkeys K_i as those are a function of h alone. We have the equivalence

$$g_N(h, x) \oplus x \oplus h = \pi_K(x \oplus N).$$

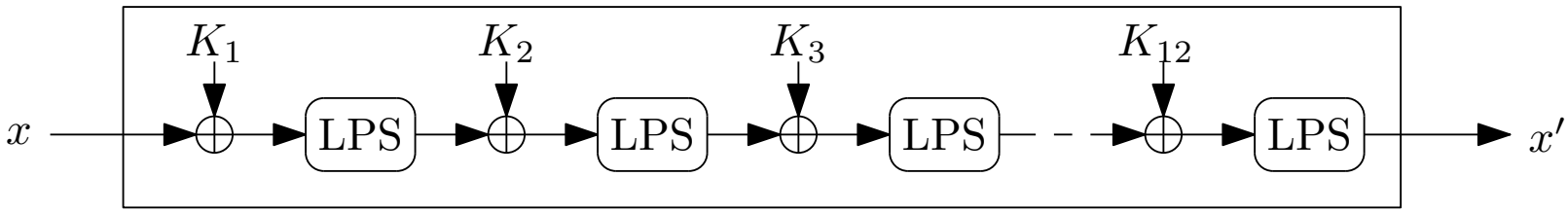
Assuming that the round constants C_i offer no advantage over known round keys K_i, π_C is as secure as π_K and any distinguisher should have the same complexity. □

We see that a generic powerful attack against π is also an attack on g. A distinguishing attack against g does not imply a collision attack against Streebog as a whole.

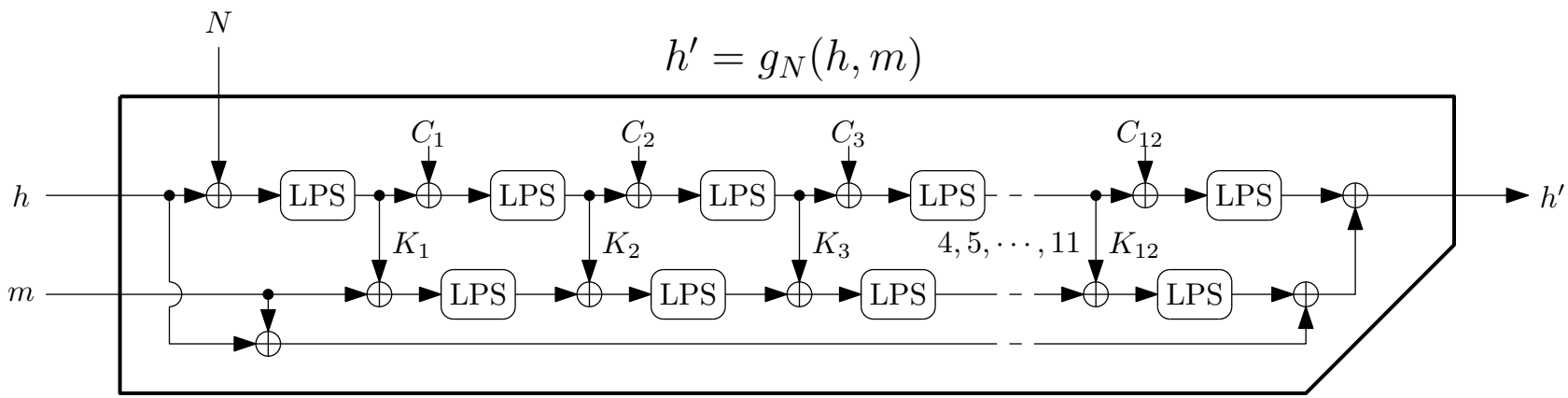
Security Reduction Explained

Just replace C with K in π:

$$x' = \pi_K(x)$$



We have $g_N(h, x) \oplus x \oplus h = \pi_K(x \oplus N)$:



StriBob Performance

StriBob requires 12 LPS invocations per 256 bits processed whereas Streebog requires 25 LPS invocations per 512 bits, making StriBob faster. Also the runtime memory requirement is cut down to 25 %.

Implementation optimization uses similar techniques as that of AES. However 64-bit rows are better suited for modern 64-bit architectures (AES is from the 90s, 32-bit era).

Algorithm	**Throughput**
AES - 128 / 192 / 256	109.2 / 90.9 / 77.9 MB/s
SHA - 256 / 512	212.7 / 328.3 MB/s
GOST 28147-89	53.3 MB/s
GOST R 34.11-1994	20.8 MB/s
GOST R 34.11-2012	109.4 MB/s
StriBob	115.7 MB/s

..as measured on my few years old Core i7 @ 2.80. My StriBob implementation, StriBob optimized reference implementation. The rest are OpenSSL (w/o AES NIS).

$3\frac{1}{2}$ "Beyond Modes: Building a Secure Record Protocol from a Cryptographic Sponge Permutation"

Modified from a paper originally presented at:
Cryptographers' Track, RSA Conference '14
San Francisco CA, February 26, 2014

Background: Complex, Insecure Legacy Protocols

All of the (de facto) standard network security protocols – SSL3, SSH2, TLS, IPSEC, PPTP, and wireless WPA2 (and its predecessors) – consist of two essentially independent protocols:

1. The handshake / authentication protocol which authenticates one or two parties and establishes a master shared secret K.
2. The transport / record protocol which provides security for application data streams based on K.

In addition to the plaintext P, data items required by record protocols to perform authenticated encryption at each direction:

K_e Key for the symmetric encryption algorithm.

K_a Key for the message authentication algorithm.

S Incremental message sequence number.

IV Initialization vector for ciphers.

That is $2 \times 4 = 8$ separate cryptovariables and at least two different algorithms (HMAC and block cipher) in addition to PRFs.

Two-party Synchronization

Legacy protocols use two independent channels: one from Alice to Bob ($A \to B$) and another from Bob to Alice ($B \to A$).

Example. Consider the following three transcripts:

$$T1: \quad B \to A : M_2, \quad A \to B : M_1, \quad A \to B : M_3$$

$$T2: \quad A \to B : M_1, \quad B \to A : M_2, \quad A \to B : M_3$$

$$T3: \quad A \to B : M_1, \quad A \to B : M_3, \quad B \to A : M_2$$

These three exchanges have precisely the same valid representation on the two channels when sent over IPSEC, TLS, SSL, or SSH protocols. The same authentication codes will match, etc.

The Synchronization Problem of Two-Channel Protocols.

Despite individual message authentication, the interwoven order of the sequence of back-and-forth messages cannot be unambiguously determined and authenticated with legacy protocols.

This is why transaction records are often authenticated on the application level as well, adding an another layer of complexity.

Issue also affects basic end-user interactive security as portions of server messaging can be maliciously delayed, encouraging the user to react to partial information.

Legal perspective on unambiguous session transcripts:
Steven J. Murdoch and Ross Anderson: "Security Protocols and Evidence: Where Many Payment Systems Fail." Financial Cryptography and Data Security 2014, 3 – 7 March 2014, Barbados.

Motivation for BLNK

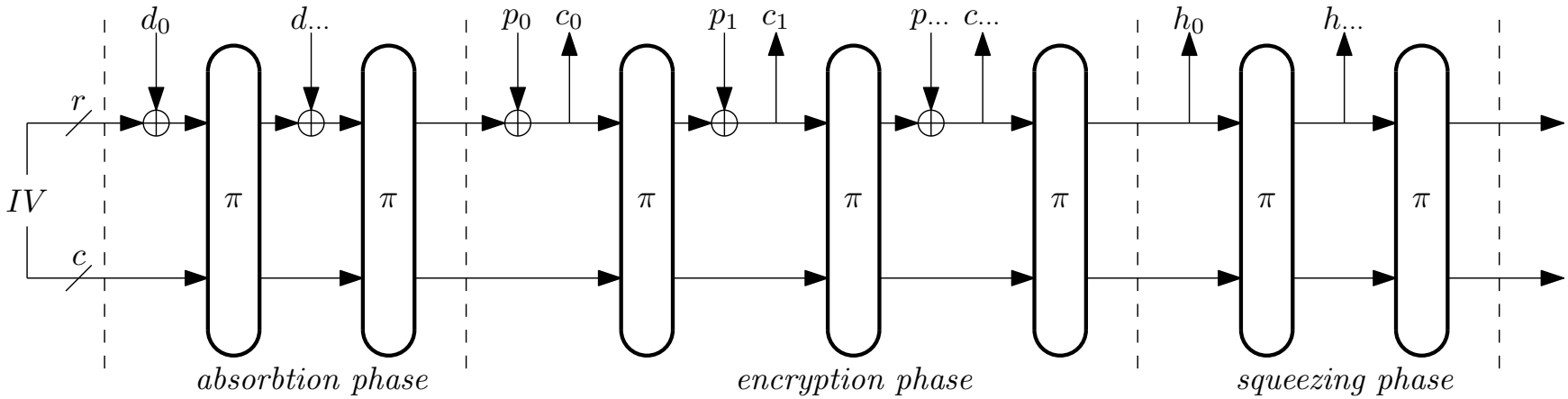
Legacy protocols are unsuited for ultra-lightweight applications.

Academic research has focused on lightweight primitives, and suitable lightweight, general purpose communications protocols have not been proposed.

We need a generic short-distance lightweight link layer security provider that can function independently from upper layer application functions.

- Design with mathematical and legal provability in mind.
- Aim at simplicity and small footprint: use a single sponge permutation for key derivation, confidentiality, integrity, etc. In traditional protocols these are all different functions.
- Use a single state variable in both directions, instead of 8+ cryptovariables. Allow forward security via *forgetting*.
- Ideally this protocol would be realizable with semi-autonomous integrated hardware, without much CPU or MCU involvement.

Recap: Sponge-based Authenticated Encryption



1. **Absorption.** Key, nonce, and associated data (d_i) are mixed.
2. **Encryption.** Plaintext p_i is used to produce ciphertext c_i.
3. **Squeezing.** Authentication Tag h_i is squeezed from the state.
4. Why not use that final state as IV for reply and go straight to Step 2 ? *(feature called "sessions" in Ketje and Keyak)*

Simplification

Legacy protocol encryption of P to C with 4 cryptovariables:

$$C = f_{\mathsf{cs}}(P, S, IV, K_e, K_a).$$

Decryption can fail in authentication (auth tag is in C):

$$f_{\mathsf{cs}}^{-1}(C, S, IV, K_e, K_a) = P \; or \; \text{FAIL}.$$

In sponges we have a state S_i, plaintext P_i, and some padding info that produces a new state and ciphertext (including a MAC):

$$(S_{i+1}, C_i) = \text{enc}(\; S_i, P_i, pad \;).$$

The decoding function dec() produces the same S_{i+1} and P_i from the ciphertext and equivalent S_i and padding, synchronizing the state between sender and receiver:

$$(S_{i+1}, P_i) = \text{dec}(\; S_i, C_i, pad \;) \; or \; \text{FAIL}.$$

Security Goals

Designers should have provable bounds on these three goals:

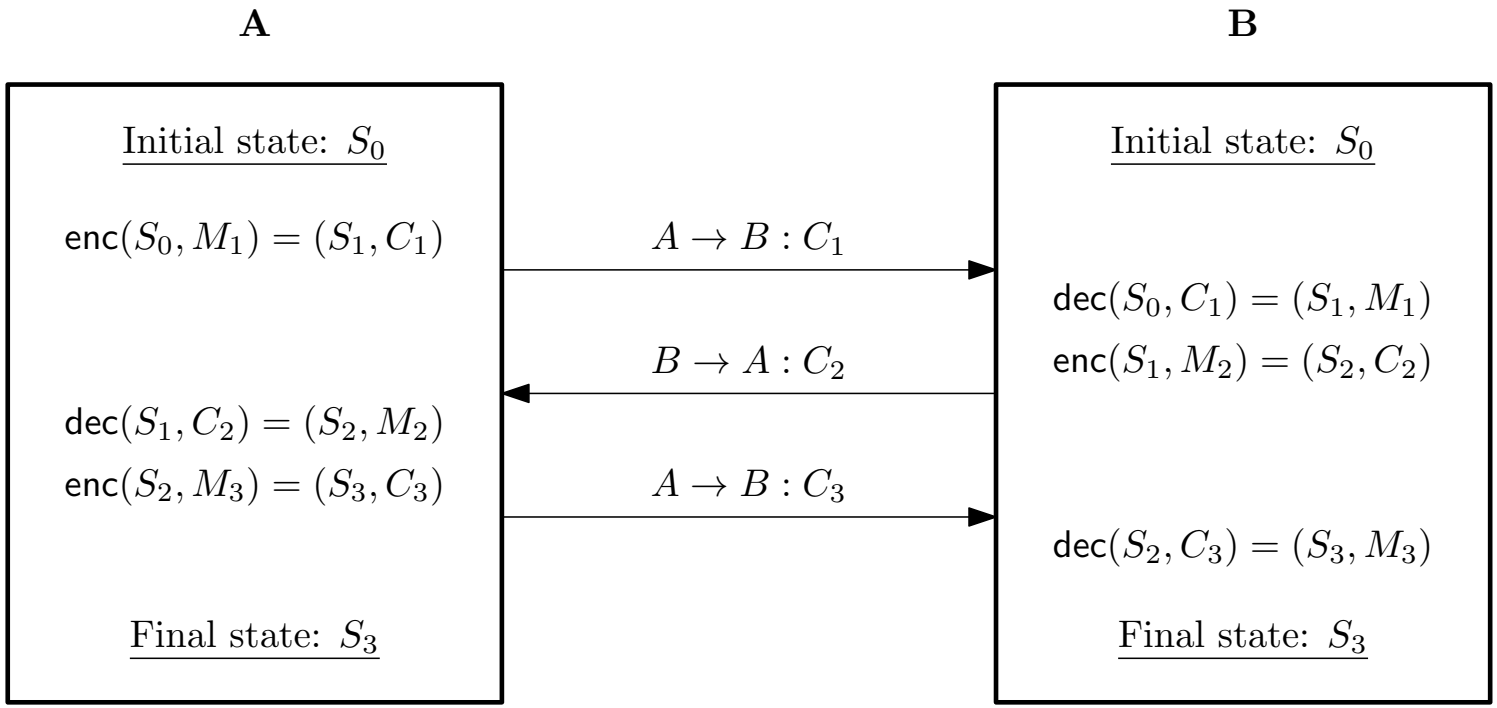
priv The ciphertext result C of $\mathrm{enc}(S, P, pad)$ must be indistinguishable from random when S is random and P may be chosen by the attacker.

auth The probability of an adversary of choosing a message C that does not result in a FAIL in $\mathrm{dec}(S, C, pad)$ without knowledge of S is bound by a function of the authentication tag size t and number of trials.

sync Each party can verify that all previous messages of the session have been correctly received and the absolute order in which messages were sent.

First two are standard Authentication Encryption requirements, the last one is new.

Solution: Just continue to use the state in reply!



Simplified interchange of three messages whose plaintext equivalents are $A \to B : M_1$, $B \to A : M_2$, $A \to B : M_3$, utilizing a synchronized secret state variables S_i.

The order of messages cannot be modified and hence this exchange is **sync**-secure !

So .. it's Half-Duplex ?

Half-duplex links may not seem ubiquitous to developers due to the use of the socket programming paradigm. Full-duplex illusion is often achieved by time-division duplexing.

- Half-duplex is physically prevalent on sensor networks, IoT and last-hop radio links: Bluetooth and IEEE 802.15.4 ZigBee are half-duplex.
- In addition to wireless last-hop transports, most RFID, smart card [ISO 7816-4, ISO 18000-63], and industrial control [MODBUS] communications are implemented under a query-response model and are therefore effectively half-duplex.
- Half-duplex links can be established wirelessly with unpaired frequencies (same frequency in both directions), or with (twisted-wire / single contact) serial links. These are a typical scenarios in lightweight time-divide communications, our specific targets.

Unambiguous Session Transcripts via Better Domain Separation

- **Keccak** only has domain separation between data input and hash output.
- **Keccak**-$160/256/512$ are distinguished from each other via different rates r, not via padding or IV; different hardware for different hash sizes?!
- **DuplexWrap** extended this to domain separation with frame bits between key material, payload data, and message authentication tag.
- **Sakura** added further frame bits yet again to facilitate tree hashing.

We want to have a extensible padding mechanism that allow same hardware to be used for any purpose.

Key feature in BLNK: originator bits; whether sponge input is from Alice, Bob, Both (e.g. DH Secret), or none in particular..

Multiplexing the Sponge

We retain one d-bit word D in S^c for domain separation; $S^c = (S^d \,||\, S^{c'})$ with $c' = c - d$. The iteration for arbitrary absorption, squeezing, and encryption is now:

$$S_{i+1} = \pi(\ S_i^r \oplus M_i \,||\, S_i^d \oplus D_i \,||\, S_i^{c'}\ ).$$

For decryption we have the following update function:

$$S_{i+1} = \pi(\ C_i \,||\, S_i^d \oplus D_i \,||\, S_i^{c'}\ ).$$

In StriBob / BLNK, $d = 8$ bits. The actual capacity suffers only by couple of bits.

Even hash and MAC outputs are padded (length padding + domain separation). This protects against length-extension attacks.

Multiplex Byte

Depending on protocol state and the intended usage of message block, multiple bits are set simultaneously. From `ref/stribob.h`:

Bit	Mask		When set
0	`BLNK_END`	`0x01`	Full input or output block (r bits)
1	`BLNK_FIN`	`0x02`	Final block of this domain element
2	`BLNK_A2B`	`0x04`	Originating from Alice
3	`BLNK_B2A`	`0x08`	Originating from Bob
4-6	`BLNK_DAT`	`0x00`	Bulk data for hashing (in)
4-6	`BLNK_KEY`	`0x10`	Secret key (in)
4-6	`BLNK_NPUB`	`0x20`	Public nonce or sequence no (in)
4-6	`BLNK_NSEC`	`0x30`	Secret nonce or sequence no (i/o)
4-6	`BLNK_AAD`	`0x40`	Authenticated Associated Data (in)
4-6	`BLNK_MSG`	`0x50`	Message Payload (i/o)
4-6	`BLNK_MAC`	`0x60`	Keyed MAC (out)
4-6	`BLNK_HASH`	`0x70`	Hash (out)
7	`BLNK_BIT7`	`0x80`	Reserved for future extensions

Example: Authenticated Record Protocol Flow (1)

We first absorb and transmit the identities I_a and I_b of Alice and Bob into the state. These are not encrypted; S_0 is IV.

Identifiers I_a and I_b should be random strings of sufficient size (at least 128 bits).

This is an optional step that helps both parties select the correct shared secret K.

$$
\begin{aligned}
(S_1, M_1) &= \mathsf{enc}(S_0, I_a, \texttt{0x44}) \mid A \to B : M_1 \\
(S_2, M_2) &= \mathsf{enc}(S_1, I_b, \texttt{0x48}) \mid B \to A : M_2 \\
S_3 &= \mathsf{enc}(S_2, K, \texttt{0x1C}) \mid \mathit{no\ transmission}
\end{aligned}
$$

K may be from a lightweight asymmetric key exchange method such as Curve25519 [Bernstein 2006] or derived from passwords.

It is never transmitted, but just absorbed in the secret state to produce S_3 from S_2.

Example: Authenticated Record Protocol Flow (2)

Two random nonces R_a and R_b are required for challenge-response authentication and to make the session unique.

$$(S_4, M_3) = \mathsf{enc}(S_3, R_a, \texttt{0x24}) \mid A \to B : M_3$$
$$(S_5, M_4) = \mathsf{enc}(S_4, R_b, \texttt{0x28}) \mid B \to A : M_4$$

We may now perform mutual authentication with tags of t bits:

$$(S_6, M_5) = \mathsf{enc}(S_5, 0^t, \texttt{0x64}) \mid A \to B : M_5$$
$$(S_7, M_6) = \mathsf{enc}(S_6, 0^t, \texttt{0x68}) \mid B \to A : M_6$$

Checking M_5 and M_6 completes mutual authentication. By an inductive process we see that the session secret S_7 is now randomized by both parties and the shared secret is not leaked.

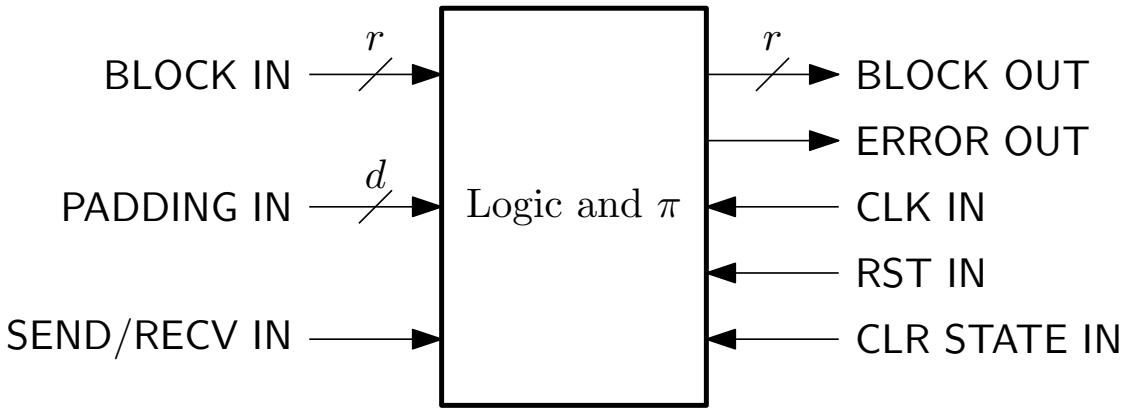
Example: Authenticated Record Protocol Flow (3)

After this, plaintexts P_a (for $A \to B$) and P_b (for $B \to A$) can be encrypted, transmitted and authenticated by repeating the following exchange:

$$
\begin{aligned}
(S_{i+1}, M_a) &= \mathsf{enc}(S_i, P_a, \texttt{0x54}) \mid A \to B : M_a \\
(S_{i+2}, T_a) &= \mathsf{enc}(S_{i+1}, 0^t, \texttt{0x64}) \mid A \to B : T_a \\
(S_{i+3}, M_b) &= \mathsf{enc}(S_{i+2}, P_b, \texttt{0x58}) \mid B \to A : M_b \\
(S_{i+4}, T_b) &= \mathsf{enc}(S_{i+3}, 0^t, \texttt{0x68}) \mid B \to A : T_b
\end{aligned}
$$

Due to explicit padding it is easy to show inductively that the entire message flow is authenticated if appropriate checks are made.

Semi-Autonomous Hardware



If we incorporate K management in the comms hardware session secrets never have to leave (and cannot leave) a specific hardware component and are inaccessible to MCU/CPU app.

Such separation is very difficult (and costly) to achieve with SSL and other legacy protocols which generally require CPU/MCU interaction to create encryption and authentication keys from session secrets.

Conclusions on BLNK mode used in STRIBOB

- Provably secure lightweight record protocols can be built from a single π permutation. No need for separte PRF, HMAC, Block Cipher, and a mode of operation. Significantly reduces implementation ROM / Flash footprint.
- Working memory required to implement the entire two-way BLNK protocol is only slightly more than b bits for the state. Legacy protocols require additional storage for two sequence counters / nonces, authenticators, cipher round keys, etc.
- Explicit padding and continuous authentication resolves synchronization issues and allows straight-forward inductive security proofs based only on a single assumption.
- Provable transcripts: final "state hash" proves the integrity of an entire transaction rather than an individual message.
- **BLNK** is a class of half-duplex security protocols. Well suited for IoT, Smart Card, RFID, NFC, and other last-lap security.

CRO of F-Secure, 29 Apr 2014 when StriCat was released.

Download StriCat Source from my web site:
`http://www.stribob.com/stricat`

- Portable, self-contained,open source, POSIX compliant, relatively small (couple of thousand lines).
- Implementation of secure links over TCP using the BLNK protocol. Can be used as a secure replacement for `netcat`.
- File encryption and decryption using an authenticated chunked file format; you can efficiently encrypt a backup stream up to terabytes in size.
- Hashing of files and streams. StriCat can also do 256- and 512-bit Streebog hashes.

Thank you for your attention!

More questions?

Finally a small demo if time and projector resolution permits.

```
$ ./stricat -h
stricat: STRIBOB / STREEBOG Cryptographic Tool.
(c) 2013-4 Markku-Juhani O. Saarinen <mjos@iki.fi>. See LICENSE.

stricat [OPTION].. [FILE]..
 -h          This help text
 -t          Quick self-test and version information

Shared secret key (use twice to verify):
 -q          Prompt for key
 -f <file>   Use file as a key
 -k <key>    Specify key on command line

Files:
 -e          Encrypt stdin or files (add .sb1 suffix)
 -d          Decrypt stdin or files (must have .sb1 suffix)
 -s          Hash stdin or files in STRIBOB BNLK mode (optionally keyed)
 -g          GOST R 34.11-2012 unkeyed Streebog hash with 256-bit output
 -G          GOST R 34.11-2012 unkeyed Streebog hash with 512-bit output

Communication via BLNK protocol:
 -p <port>   Specify TCP port (default 48879)
 -c <host>   Connect to a specific host (client)
 -l          Listen to incoming connection (server)
```