Beyond Modes: Building a Secure Record Protocol from a Cryptographic Sponge Permutation

Author: Markku-Juhani O. Saarinen

Presented by: Jean-Philippe Aumasson

KUDELSKI SECURITY

CT-RSA '14, San Francisco, USA
26 February 2014

Background: Complex, Insecure Legacy Protocols

All of the RFC / de facto standard networking security protocols—SSL3, SSH2, TLS, IPSEC, PPTP, and wireless WPA2 (together with its predecessors)—consist of two largely independent protocols:

1. The **handshake / authentication protocol** which establishes a shared secret K.
2. The **transport / record protocol** which provides communications security.

Background: Complex, Insecure Legacy Protocols

All of the RFC / de facto standard networking security protocols—SSL3, SSH2, TLS, IPSEC, PPTP, and wireless WPA2 (together with its predecessors)—consist of two largely independent protocols:

1. The **handshake / authentication protocol** which establishes a shared secret K.
2. The **transport / record protocol** which provides communications security.

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

- S Incremental message sequence number.
- IV Initialization vector for block ciphers.
- K_e Key for the symmetric encryption algorithm.
- K_a Key for the message authentication algorithm.

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

Motivation for BLINKER

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.

Motivation for BLINKER

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. (Instead of distinct algorithms.)
- Use a **single state variable** in both directions, instead of 8+ cryptovariables.
- Ideally this protocol would be realizable with semi-autonomous integrated hardware, without much CPU or MCU involvement.

Two-party Synchronization

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

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:

$$
\begin{aligned}
T1: &\quad B \to A : M_2, \;\; A \to B : M_1, \;\; A \to B : M_3 \\
T2: &\quad A \to B : M_1, \;\; B \to A : M_2, \;\; A \to B : M_3 \\
T3: &\quad A \to B : M_1, \;\; A \to B : M_3, \;\; B \to A : M_2
\end{aligned}
$$

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.

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.

The Synchronization Problem of Two-Channel Protocols.

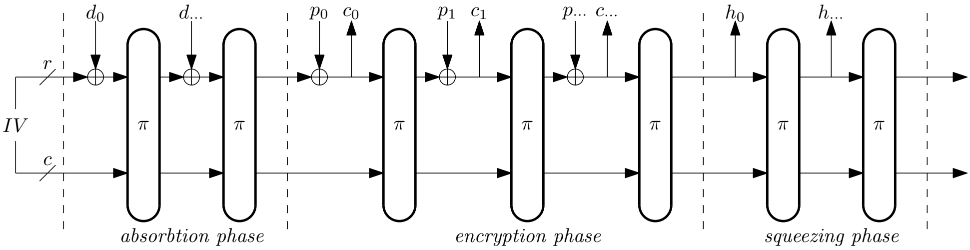
Despite individual message authentication, the **interwoven order** of the sequence of back-and-forth messages cannot be unambiguously determined and authenticated with legacy protocols.

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

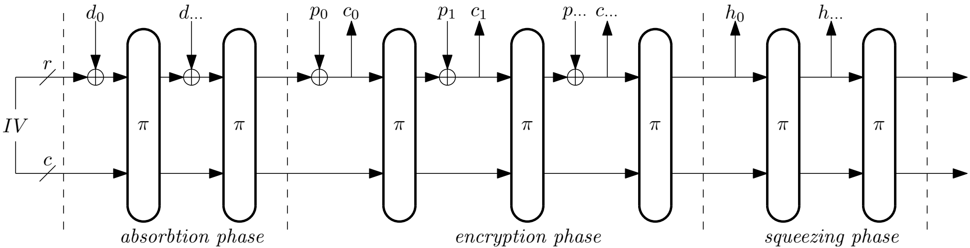
Issue also affects basic **end-user interactive security** as portions of server messaging can be maliciously delayed, encouraging the user to react to partial information.

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.

Recap: Sponge-based Authenticated Encryption



Recap: Sponge-based Authenticated Encryption



1. **Absorption.** Key, nonce, and associated data (d_i) are mixed into the state.
2. **Encryption.** Plaintext p_i is used to produce ciphertext c_i (or vice versa).
3. **Squeezing.** Message 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 ?

Simplification

Legacy protocol encryption of P to C with 4 cryptovariables:

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

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):

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

Simplification

Legacy protocol encryption of P to C with 4 cryptovariables:

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

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

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

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):

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

Simplification

Legacy protocol encryption of P to C with 4 cryptovariables:

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

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

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

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):

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

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:

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

Security Goals

Protocol designers should have provable bounds on these three goals:

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

Security Goals

Protocol 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.

Security Goals

Protocol designers should have provable bounds on these three goals:

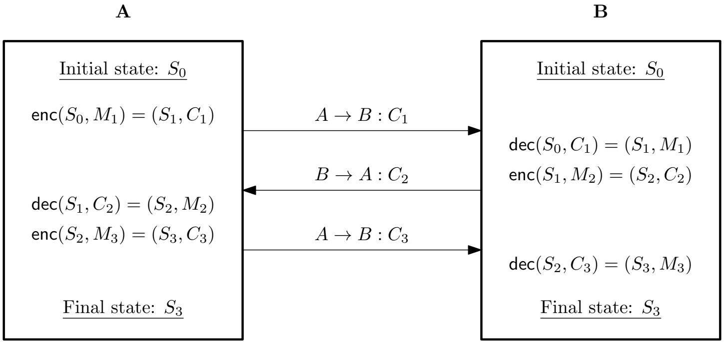
priv The ciphertext result C of $\mathrm{enc}(S, P, pad)$ must be indistinguishable from random when S is random and P may be chosen by the attacker.

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.

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.

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.

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?!

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?!
- **SpongeWrap** extended this to domain separation with *frame bits* between key material, payload data, and message authentication tag.

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?!
- **SpongeWrap** 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.

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?!
- **SpongeWrap** 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**.

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?!
- **SpongeWrap** 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 BLINKER: **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**; $S^c = (S^d \mid\mid S^{c'})$ with $c' = c - d$. The iteration for arbitrary absorption, squeezing, and encryption is now:

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

Multiplexing the Sponge

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

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

For decryption we have the following update function:

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

Multiplexing the Sponge

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

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

For decryption we have the following update function:

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

In BLINKER, $d = 16$ bits. We estimate that the capacity suffers only by few bits.

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

Multiplex Word

Depending on protocol state and the intended usage of message block, multiple bits are set simultaneously. Here's an example set:

Bit	Mask	When set
0	0x0001	This is a full input or output block (r bits).
1	0x0002	This is the final block of this data element.
4	0x0004	Block is an input to sponge ("absorption").
3	0x0008	Block is output from sponge ("squeezing").
4	0x0010	Associated Authenticated Data (in).
5	0x0020	Secret key (in).
6	0x0040	Nonce or sequence number (in).
7	0x0080	Encryption / Decryption (in and out).
8	0x0100	Hash block (out).
9	0x0200	Keyed Message Authentication Code (MAC) (out).
10	0x0400	Block for state storage or reloading (in or out).
11	0x0800	Pseudo Random Number Generator (PRNG) (feed or out).
12	0x1000	Originating from Alice – client / slave.
13	0x2000	Originating from Bob – server / master.
14	0x4000	Tree chaining Node.
15	0x8000	Tree final Node.

Example: Authentication and 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 as S_0 is the Initialization Vector.
We recommend identifiers I_a and $I_b$to be random strings of sufficient size (at least 128 bits).

Example: Authentication and 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 as S_0 is the Initialization Vector.
We recommend identifiers I_a and $I_b$to 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.

$$
\begin{aligned}
(S_1, M_1) &= \mathrm{enc}(S_0, I_a, \texttt{0x108C}) \mid A \rightarrow B : M_1 \\
(S_2, M_2) &= \mathrm{enc}(S_1, I_b, \texttt{0x208C}) \mid B \rightarrow A : M_2 \\
S_3 &= \mathrm{enc}(S_2, K, \texttt{0x3024}) \mid no\ transmission
\end{aligned}
$$

Example: Authentication and 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 as S_0 is the Initialization Vector.
We recommend identifiers I_a and $I_b$to 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.

$$
\begin{aligned}
(S_1, M_1) &= \mathsf{enc}(S_0, I_a, \mathtt{0x108C}) \mid A \to B : M_1 \\
(S_2, M_2) &= \mathsf{enc}(S_1, I_b, \mathtt{0x208C}) \mid B \to A : M_2 \\
S_3 &= \mathsf{enc}(S_2, K, \mathtt{0x3024}) \mid no\ transmission
\end{aligned}
$$

K may be derived with 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: Authentication and Record Protocol Flow (2)

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

$$(S_4, M_3) = \mathrm{enc}(S_3, R_a, \texttt{0x10CC}) \mid A \to B : M_3$$
$$(S_5, M_4) = \mathrm{enc}(S_4, R_b, \texttt{0x20CC}) \mid B \to A : M_4$$

Example: Authentication and Record Protocol Flow (2)

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

$$(S_4, M_3) = \mathrm{enc}(S_3, R_a, \mathtt{0x10CC}) \mid A \rightarrow B : M_3$$
$$(S_5, M_4) = \mathrm{enc}(S_4, R_b, \mathtt{0x20CC}) \mid B \rightarrow A : M_4$$

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

$$(S_6, M_5) = \mathrm{enc}(S_5, 0^t, \mathtt{0x1208}) \mid A \rightarrow B : M_5$$
$$(S_7, M_6) = \mathrm{enc}(S_6, 0^t, \mathtt{0x2208}) \mid B \rightarrow A : M_6$$

Checking M_5 and M_6 completes mutual authentication. By an inductive process we see that the session secret S_7 is now dependent upon randomizers from both parties and the original shared secret is not leaked if the sponge satisfies our security axioms.

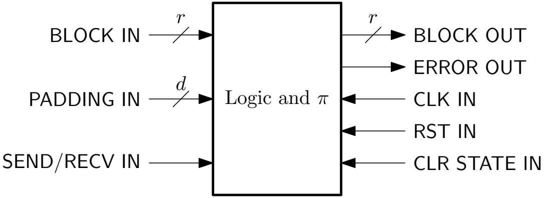
Example: Authentication and Record Protocol Flow (3)

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

$$\begin{aligned}
(S_{i+1}, M_a) &= \mathrm{enc}(S_i, P_a, \mathtt{0x108C}) \mid A \rightarrow B : M_a \\
(S_{i+2}, T_a) &= \mathrm{enc}(S_{i+1}, 0^t, \mathtt{0x1208}) \mid A \rightarrow B : T_a \\
(S_{i+3}, M_b) &= \mathrm{enc}(S_{i+2}, P_b, \mathtt{0x208C}) \mid B \rightarrow A : M_b \\
(S_{i+4}, T_b) &= \mathrm{enc}(S_{i+3}, 0^t, \mathtt{0x2208}) \mid B \rightarrow A : T_b
\end{aligned}$$

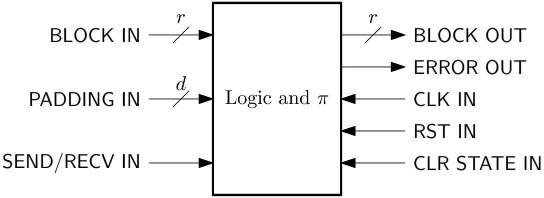
Due to explicit padding it is easy to show **inductively** that the entire message flow is authenticated if appropriate checks are made.

Semi-Autonomous Hardware and Lightweight Demo Software



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.

Semi-Autonomous Hardware and Lightweight Demo Software



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.

cblnk.com
https://www.cblnk.com/cb0cat/
cblnk.com
KUDELSKI SECURITY
cb0cat 0.20131216
1. Introduction and License
2. Download, Compile, and Test
3. Hashing
4. Keying
5. Encryption and Decryption
6. Networking and File Transfer
7. Binding a Shell or Command
1. Introduction and License
This is a quick tutorial to the cb0cat multi-use cryptographic tool, which can be used to hash, encrypt, and decrypt files and to establish secure communication links over TCP. cb0cat has been designed to be self-contained, portable, and extremely lightweight (currently only about 1500 lines).

Conclusions

- **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.

Conclusions

- **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 BLINKER 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.

Conclusions

- **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 BLINKER 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.

Conclusions

- **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 BLINKER 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.

Conclusions

- **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 BLINKER 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.
- **BLINKER**: a class of lightweight half-duplex protocols. Especially suited for IoT, Smart Card, RFID, NFC, and other last-lap security.