Symmetric Key Exchange: Lightweight Alternatives for a Post-Quantum IoT

Bor de Kock
Assistant Professor of Cryptology at NTNU Trondheim
Symmetric Key Exchange: Lightweight Alternatives for a Post-Quantum IoT

Bor de Kock

Post-Quantum Cryptography Conference
November 8, 2023
About me

Bor de Kock, assistant professor cryptology
• MSc from Eindhoven University of Technology
• PhD on Post-Quantum Key Exchange, from NTNU

My research interests:
Post-quantum cryptography, key exchange, password-based crypto, authentication, security models, ratcheting, etc.

*in other words: practical crypto!*
Norwegian University of Science and Technology
Trondheim, Gjøvik and Ålesund
Largest university in Norway
43,000 students
32 cryptographers
Let’s talk key exchange…
Let’s talk key exchange…

• Making key exchange post-quantum is an ongoing effort
Let’s talk key exchange…

• Making key exchange post-quantum is an ongoing effort
• Most serious candidates are inefficient, compared to SoA
Let’s talk key exchange…

- Making key exchange post-quantum is an ongoing effort
- Most serious candidates are inefficient, compared to SoA
Let’s talk key exchange…

• Making key exchange post-quantum is an ongoing effort
• Most serious candidates are inefficient, compared to SoA
• Symmetric algorithms such as AES are post-quantum
Let’s talk key exchange…

• Making key exchange post-quantum is an ongoing effort
• Most serious candidates are inefficient, compared to SoA

• Symmetric algorithms such as AES are post-quantum
• Symmetric algorithms such as AES are very efficient
Let’s talk key exchange…

• Making key exchange post-quantum is an ongoing effort
• Most serious candidates are inefficient, compared to SoA

• Symmetric algorithms such as AES are post-quantum
• Symmetric algorithms such as AES are very efficient
• Many security features we like are missing.
What do we want to achieve?
What do we want to achieve?

• Authenticated key exchange for very constrained devices
What do we want to achieve?

• Authenticated key exchange for very constrained devices
• Pre-shared symmetric keys
What do we want to achieve?

- Authenticated key exchange for very constrained devices
- Pre-shared symmetric keys
- Forward security
What do we want to achieve?

• Authenticated key exchange for very constrained devices
• Pre-shared symmetric keys
• Forward security
• Synchronization
What do we want to achieve?

• Authenticated key exchange for very constrained devices
• Pre-shared symmetric keys
• Forward security
• Synchronization
• Concurrent Correctness
In this talk…

Symmetric Key Exchange with Full Forward Security and Robust Synchronization

Colin Boyd, Gareth T. Davies, Bor de Kock, Kai Gellert, Tibor Jager and Lise Millerjord
In this talk…

Symmetric Key Exchange with Full Forward Security and Robust Synchronization
Colin Boyd, Gareth T. Davies, Bor de Kock, Kai Gellert, Tibor Jager and Lise Millerjord

• 3 very efficient AKE protocols with linear key evolution
In this talk…

Symmetric Key Exchange with Full Forward Security and Robust Synchronization

Colin Boyd, Gareth T. Davies, Bor de Kock, Kai Gellert, Tibor Jager and Lise Millerjord

- 3 very efficient AKE protocols with linear key evolution
- 2 AKE protocols with non-linear key evolution
In this talk…

Symmetric Key Exchange with Full Forward Security and Robust Synchronization

Colin Boyd, Gareth T. Davies, Bor de Kock, Kai Gellert, Tibor Jager and Lise Millerjord

• 3 very efficient AKE protocols with linear key evolution
• 2 AKE protocols with non-linear key evolution
• Framework for protocol analysis
In this talk…

*Symmetric Key Exchange with Full Forward Security and Robust Synchronization*

Colin Boyd, Gareth T. Davies, Bor de Kock, Kai Gellert, Tibor Jager and Lise Millerjord

- 3 very efficient AKE protocols with linear key evolution
- 2 AKE protocols with non-linear key evolution
- Framework for protocol analysis
- Formalization of synchronization robustness as a security property
Authenticated Key Exchange - AKE
Authenticated Key Exchange - AKE
Authenticated Key Exchange - AKE

Alice

Bob

$K_{AB}$

$K_{AB}$
Authenticated Key Exchange - AKE

Alice

Bob

$K_{AB}$

$K_{AB}$
Authenticated Key Exchange - AKE

Alice

Bob

$K_{AB}$

$K_S$

$K_{AB}$

$K_S$
Forward Security
Forward Security

Alice

Bob

$K_{AB}$
Forward Security

Alice

Bob

\[ K_{AB} \]

\[ K_{S1} \]

\[ K_{AB} \]

\[ K_{S1} \]
Forward Security

Alice

Bob

$K_{AB}$

$K_{S1}$

$K_{S2}$

$K_{AB}$

$K_{S1}$

$K_{S2}$
Forward Security

Bob

Alice

K_{AB}

K_{S1}

K_{S2}

Bob

K_{AB}

K_{S1}

K_{S2}
Forward Security

Alice

Bob

\[ K_{AB} \]

\[ K_{S1} \]

\[ K_{S2} \]

\[ K_{AB} \]

\[ K_{S1} \]

\[ K_{S2} \]
Forward Security

Alice

Bob

\(K_{AB}\)

\(K_{S1}\)

\(K_{S2}\)
Forward Security

Alice

Bob

$K_{AB}$

$K_{S1}$

$K_{S2}$

$K_{S1}$

$K_{S2}$
Achieving Forward Security
Achieving Forward Security

• Evolve keys to obtain forward security
Achieving Forward Security

• Evolve keys to obtain forward security
• Time-based evolution [Dousti and Jalili, 2015]
Achieving Forward Security

• Evolve keys to obtain forward security
• Time-based evolution [Dousti and Jalili, 2015]
• Triggered evolution: evolve after session key derivation
Challenges
Challenges

• Synchronization - both parties needs to have evolved the same number of steps
Challenges

• Synchronization - both parties needs to have evolved the same number of steps
• Concurrent correctness – parallel sessions cause problems when one session evolves shared key material before the other session is ready
Challenges

- Synchronization - both parties needs to have evolved the same number of steps
- Concurrent correctness – parallel sessions cause problems when one session evolves shared key material before the other session is ready

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SAKE [ACF20]</td>
<td>Mutual</td>
<td>5</td>
<td>❌</td>
<td>❌</td>
<td>✔</td>
</tr>
<tr>
<td>SAKE-AM [ACF20]</td>
<td>Mutual</td>
<td>4</td>
<td>❌</td>
<td>❌</td>
<td>✔</td>
</tr>
</tbody>
</table>
Example protocol: LP2
Example protocol: LP2
Example protocol: LP2
Example protocol: LP2

CTR = 0

Alice

CTR = 0

Bob
Example protocol: LP2

CTR = 0

CTR = 0
Example protocol: LP2

Alice

CTR = 0

CTR = 1

Bob

CTR = 0

MAC

$K_M$, $K_{AB}$
Example protocol: LP2

CTR = 0
CTR = 1

Accept?
Example protocol: LP2

CTR = 0
CTR = 0
Accept?
CTR = 1

CTR = 1

Alice

Bob
Example protocol: LP2

Alice

\(K_M\)

\(K_{AB}\)

CTR = 0

CTR = 1

Bob

\(K_M\)

\(K_{AB}\)

CTR = 0

Accept?

CTR = 1

\(K_S \leftarrow \text{Key-derivation}(K_{AB})\)

\(K_{AB} \leftarrow \text{Evolve}(K_{AB})\)

CTR = CTR + 1 = 2
Example protocol: LP2

CTR = 0
Accept?

CTR = 1

MAC

CTR = 0
Accept?

CTR = 1

MAC

$K_M$, $K_{AB}$

Accept?
Example protocol: LP2

**Alice**

CTR = 0

CTR = 1

Accept?

\[ K_S \leftarrow \text{Key-derivation}(K_{AB}) \]

\[ K_{AB} \leftarrow \text{Evolve}(K_{AB}) \]

CTR = CTR + 1 = 2

**Bob**

CTR = 0

Accept?

CTR = 1

\[ K_S \leftarrow \text{Key-derivation}(K_{AB}) \]

\[ K_{AB} \leftarrow \text{Evolve}(K_{AB}) \]

CTR = CTR + 1 = 2
Example protocol: LP2

CTR = 2

Alice

Bob
Example protocol: LP2

CTR = 2
CTR = 3

\( K_M \) \( K_{AB} \)

Bob

CTR = 2
Example protocol: LP2

Alchemy

CTR = 2
CTR = 3
Accept?

Bob

CTR = 2

Alice

K_M, K_AB

000

K_M, K_AB

000
Example protocol: LP2

- **CTR = 2**
- **CTR = 3**
- **CTR = 4**

**Alice**
- $K_M$
- $K_{AB}$
- 000

**Bob**
- $K_M$
- $K_{AB}$
- 000

Accept?

CTR = 3

Derive session key
Evolve $K_{AB}$

CTR = 4
Example protocol: LP2

Alice

CTR = 2
CTR = 3

Bob

CTR = 2
Accept?
CTR = 3
Derive session key
Evolve $K_{AB}$
CTR = 4

$K_{M}$
$K_{AB}$

$000$

$000$
Example protocol: LP2

Alice

CTR = 2
CTR = 3

Bob

CTR = 2
Accept?
CTR = 3
Derive session key
Evolve $K_{AB}$
CTR = 4

Next session:
CTR = 5

$K_M$, $K_{AB}$
Linear-evolving protocols
Linear-evolving protocols

- 3 protocols – 1, 2 and 3 messages
Linear-evolving protocols

- 3 protocols – 1, 2 and 3 messages
- 1-message protocol: one-way authentication
Linear-evolving protocols

- 3 protocols – 1, 2 and 3 messages
- 1-message protocol: one-way authentication
- 3-message protocol: key confirmation, bounded gap
Linear-evolving protocols

- 3 protocols – 1, 2 and 3 messages
- 1-message protocol: one-way authentication
- 3-message protocol: key confirmation, bounded gap

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Weak</td>
<td>Weak</td>
<td>Weak</td>
<td></td>
</tr>
<tr>
<td>LP1</td>
<td>R only</td>
<td>1</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
<td>✓</td>
</tr>
<tr>
<td>LP2</td>
<td>Mutual</td>
<td>2</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
<td>✓</td>
</tr>
<tr>
<td>LP3</td>
<td>Mutual</td>
<td>3</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
<td>✓</td>
</tr>
</tbody>
</table>
Security model
Security model

• Framework for protocol analysis
Security model

• Framework for protocol analysis
• AKE model from [Bellare Rogaway 94, Li et al 2014]
Security model

• Framework for protocol analysis
• AKE model from [Bellare Rogaway 94, Li et al 2014]
• Model lacks notion of concurrent correctness and synchronization
Security model

• Framework for protocol analysis
• AKE model from [Bellare Rogaway 94, Li et al 2014]
• Model lacks notion of concurrent correctness and synchronization
• Formalization of synchronization robustness – the ability to compute keys in future sessions if something goes wrong
Concurrent Correctness

Alice

Bob

$K_M$ $K_{AB}$

CTR = 0

CTR = 0
Concurrent Correctness

Initiate session 1: CTR = 1
Concurrent Correctness

Initiate session 1: $CTR = 1$

Accept?

$CTR = 1$, session completes, Bob accepts with $CTR = 2$
Concurrent Correctness

Initiate session 1: CTR = 1
Initiate session 2: CTR = 3

CTR = 1, session completes, Bob accepts with CTR = 2
Concurrent Correctness

Initiate session 1: CTR = 1
Initiate session 2: CTR = 3
Accept?
Alice initiates session 1: CTR = 1
Accept?
Bob accepts with CTR = 2
Accept?
CTR = 1, session completes, Bob accepts with CTR = 2
Alice is at CTR = 3, aborts session 1

CTR = 0
CTR = 0
CTR = 1
CTR = 3
CTR = 1
Concurrent Correctness

**Alice**

- Initial CTR = 0

  - Initiate session 1: CTR = 1
  - Accept?
  - CTR = 1, session completes,
    Bob accepts with CTR = 2
  - CTR = 3

**Bob**

- Initial CTR = 0

  - Accept?
  - CTR = 1, session completes,
    Bob accepts with CTR = 2
  - CTR = 3

Accept?

- CTR = 3, session completes,
  Bob accepts with CTR = 4

Alice is at CTR = 3, aborts session 1
Concurrent Correctness

Alice

CTR = 0

\[000\]

Initiate session 1: CTR = 1
Accept?

Initiate session 2: CTR = 3
Accept?

Alice is at CTR = 3,
Aborts session 1
Accept?

Alice accepts with CTR = 4

Bob

CTR = 0

\[000\]

Accept?

CTR = 1, session completes,
Bob accepts with CTR = 2
Accept?

CTR = 3, session completes,
Bob accepts with CTR = 4
Synchronization Robustness
Synchronization Robustness

- Captures the ability of two parties succeeding in exchanging a session key in the future, no matter what has happened previously.
Synchronization Robustness

• Captures the ability of two parties succeeding in exchanging a session key in the future, no matter what has happened previously.

• If the parties get out of sync, we need to be able to resynchronize.
Synchronization Robustness

- Captures the ability of two parties succeeding in exchanging a session key in the future, no matter what has happened previously.
- If the parties get out of sync, we need to be able to resynchronize.
- This definition formalizes this requirement and comes in a weak and a strong flavour.
Weak Synchronization Robustness
Weak Synchronization Robustness

• Definition: Any honestly executed, uninterrupted session will succeed no matter what has happened before.
Weak Synchronization Robustness

• Definition: Any honestly executed, uninterrupted session will succeed no matter what has happened before.
  – Concurrent sessions were initiated
Weak Synchronization Robustness

• Definition: Any honestly executed, uninterrupted session will succeed no matter what has happened before.
  – Concurrent sessions were initiated
  – Messages in previous sessions were dropped, reordered or altered (so that they were not accepted)
Weak Synchronization Robustness

• Definition: Any honestly executed, uninterrupted session will succeed no matter what has happened before.
  – Concurrent sessions were initiated
  – Messages in previous sessions were dropped, reordered or altered (so that they were not accepted)
  – Parties are arbitrarily many steps out of sync
Weak Synchronization Robustness

• Definition: Any honestly executed, uninterrupted session will succeed no matter what has happened before.
  – Concurrent sessions were initiated
  – Messages in previous sessions were dropped, reordered or altered (so that they were not accepted)
  – Parties are arbitrarily many steps out of sync
  – Either way: the next session Alice and Bob are allowed to execute without any interruption will succeed
Weak Synchronization Robustness

• Definition: Any honestly executed, uninterrupted session will succeed no matter what has happened before.
  – Concurrent sessions were initiated
  – Messages in previous sessions were dropped, reordered or altered (so that they were not accepted)
  – Parties are arbitrarily many steps out of sync
  – Either way: the next session Alice and Bob are allowed to execute without any interruption will succeed

• LP2: Allowing role reversal will make the protocol fail to meet this requirement
Full Synchronization Robustness
Full Synchronization Robustness

• Definition: Any honestly executed session will succeed, no matter what else is going on with concurrent sessions or previous sessions.
Full Synchronization Robustness

• Definition: Any honestly executed session will succeed, no matter what else is going on with concurrent sessions or previous sessions.
  – Arbitrary many concurrent sessions are allowed
Full Synchronization Robustness

• Definition: Any honestly executed session will succeed, no matter what else is going on with concurrent sessions or previous sessions.
  – Arbitrary many concurrent sessions are allowed
  – The adversary may interleave messages with concurrent sessions arbitrarily
Full Synchronization Robustness

- Definition: Any honestly executed session will succeed, no matter what else is going on with concurrent sessions or previous sessions.
  - Arbitrary many concurrent sessions are allowed
  - The adversary may interleave messages with concurrent sessions arbitrarily
  - The adversary may make any previous or concurrent sessions fail, but the one that is allowed to complete honestly will succeed
Full Synchronization Robustness

- Definition: Any honestly executed session will succeed, no matter what else is going on with concurrent sessions or previous sessions.
  - Arbitrary many concurrent sessions are allowed
  - The adversary may interleave messages with concurrent sessions arbitrarily
  - The adversary may make any previous or concurrent sessions fail, but the one that is allowed to complete honestly will succeed

- Linearly evolving protocols fail this requirement
Non-linear key evolution
Non-linear key evolution

- Need something different to achieve full synchronization robustness
Non-linear key evolution

- Need something different to achieve full synchronization robustness
- Use puncturable pseudorandom functions [Sahai Waters 2014]
Non-linear key evolution

- Need something different to achieve full synchronization robustness
- Use puncturable pseudorandom functions [Sahai Waters 2014]
- Definition: A PPRF is a PRF with an extra algorithm $\text{PUNCT}(k, x)$ such that
  - Evaluating on a punctured value fails
  - Puncturing on an already punctured value returns the same key
  - Puncturing is commutative – the order in which you puncture values does not matter
Non-linear key evolution

• Need something different to achieve full synchronization robustness
• Use puncturable pseudorandom functions [Sahai Waters 2014]
• Definition: A PPRF is a PRF with an extra algorithm \( \text{PUNCT}(k, x) \) such that
  – Evaluating on a punctured value fails
  – Puncturing on an already punctured value returns the same key
  – Puncturing is commutative – the order in which you puncture values does not matter
• Session key is determined by evaluating on the session nonce
Non-linear key evolution

- Need something different to achieve full synchronization robustness
- Use puncturable pseudorandom functions [Sahai Waters 2014]
- Definition: A PPRF is a PRF with an extra algorithm \( \text{PUNCT}(k, x) \) such that
  - Evaluating on a punctured value fails
  - Puncturing on an already punctured value returns the same key
  - Puncturing is commutative – the order in which you puncture values does not matter
- Session key is determined by evaluating on the session nonce
- All concurrent sessions can succeed: puncturing only affects key material of that particular session
# The non-linearly evolving protocols

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PP1</td>
<td>R only</td>
<td>1</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>PP2</td>
<td>Mutual</td>
<td>2</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
# All our protocols

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SAKE [ACF20]</td>
<td>Mutual</td>
<td>5</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>✓</td>
</tr>
<tr>
<td>SAKE-AM [ACF20]</td>
<td>Mutual</td>
<td>4</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>✓</td>
</tr>
<tr>
<td>LP1</td>
<td>R only</td>
<td>1</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>LP2</td>
<td>Mutual</td>
<td>2</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>LP3</td>
<td>Mutual</td>
<td>3</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>PP1</td>
<td>R only</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PP2</td>
<td>Mutual</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Concluding…
Concluding…

- Symmetric cryptography: it’s more relevant than you think
Concluding...

- Symmetric cryptography: it’s more relevant than you think
- Post-Quantum ≠ Key Exchange and Signatures
Concluding...

- Symmetric cryptography: it’s more relevant than you think
- Post-Quantum ≠ Key Exchange and Signatures
- We need to rethink our systems, not just our protocols
Concluding…

• Symmetric cryptography: it’s more relevant than you think
• Post-Quantum ≠ Key Exchange and Signatures
• We need to rethink our systems, not just our protocols
Concluding...

• Symmetric cryptography: it’s more relevant than you think
• Post-Quantum ≠ Key Exchange and Signatures
• We need to rethink our systems, not just our protocols

With regard to our work...
Concluding…

• Symmetric cryptography: it’s more relevant than you think
• Post-Quantum ≠ Key Exchange and Signatures
• We need to rethink our systems, not just our protocols

With regard to our work…

• Implementation efforts are underway
Concluding…

- Symmetric cryptography: it’s more relevant than you think
- Post-Quantum ≠ Key Exchange and Signatures
- We need to rethink our systems, not just our protocols

With regard to our work…

- Implementation efforts are underway
- No real world test data yet, but theoretical analysis promising
Concluding...

- Symmetric cryptography: it’s more relevant than you think
- Post-Quantum ≠ Key Exchange and Signatures
- We need to rethink our systems, not just our protocols

With regard to our work...

- Implementation efforts are underway
- No real world test data yet, but theoretical analysis promising
- Let me know if you want to get involved!
More info?

Symmetric Key Exchange with Full Forward Security and Robust Synchronization

Colin Boyd, Gareth T. Davies, Bor de Kock, Kai Gellert, Tibor Jager and Lise Millerjord

IACR ePrint 2021 / 702

bor.dekock@ntnu.no

@bordekock