Generic Compilers for Authenticated Key Exchange ASIACRYPT '10

Tibor Jager, Florian Kohlar, Sven Schäge, Jörg Schwenk
Horst Görtz Institute for IT-Security
Ruhr-Universität Bochum, Germany

12/07/2010

RUB

Overview

1. Motivation & Introduction
2. Compiler 1: KE + DSIG → AKE
3. Compiler 2: KE + A → AKE
4. Conclusion

Motivation

"Despite the importance of proofs in assuring protocol implementers about the security properties of key establishment protocols, many protocol designers fail to provide any proof of security." [CBH06]

There is a problem with applied (A)KE protocols today

- Many provably secure protocols for key exchange (KE) and authentication (A) are not used in practice ...
- ... and many practical protocols have not been proven to be secure

Motivation

"Despite the importance of proofs in assuring protocol implementers about the security properties of key establishment protocols, many protocol designers fail to provide any proof of security." [CBH06]

There is a problem with applied (A)KE protocols today

- Many provably secure protocols for key exchange (KE) and authentication (A) are not used in practice ...
- ... and many practical protocols have not been proven to be secure

Motivation

"Despite the importance of proofs in assuring protocol implementers about the security properties of key establishment protocols, many protocol designers fail to provide any proof of security." [CBH06]

There is a problem with applied (A)KE protocols today

- Many provably secure protocols for key exchange (KE) and authentication (A) are not used in practice ...
- ... and many practical protocols have not been proven to be secure

Motivation II

To solve this problem we have two choices:

Straightforward Solution 1

- Enforce the use of secure (AKE) protocols in practice

Straightforward Solution 2

- Proof the security of real-world protocols (e.g. TLS)

Our solution

- Take a real-world protocol (e.g. TLS) while only requiring minimum security properties and ...
- construct a compiler such that the resulting protocol meets the (much stronger) standard security notions

Motivation II

To solve this problem we have two choices:

Straightforward Solution 1

- Enforce the use of secure (AKE) protocols in practice

Straightforward Solution 2

- Proof the security of real-world protocols (e.g. TLS)

Our solution

- Take a real-world protocol (e.g. TLS) while only requiring minimum security properties and ...
- construct a compiler such that the resulting protocol meets the (much stronger) standard security notions

Motivation II

To solve this problem we have two choices:

Straightforward Solution 1

- Enforce the use of secure (AKE) protocols in practice

Straightforward Solution 2

- Proof the security of real-world protocols (e.g. TLS)

Our solution

- Take a real-world protocol (e.g. TLS) while only requiring minimum security properties and ...
- construct a compiler such that the resulting protocol meets the (much stronger) standard security notions

Motivation II

To solve this problem we have two choices:

Straightforward Solution 1

- Enforce the use of secure (AKE) protocols in practice

Straightforward Solution 2

- Proof the security of real-world protocols (e.g. TLS)

Our solution

- Take a real-world protocol (e.g. TLS) while only requiring minimum security properties and ...
- construct a compiler such that the resulting protocol meets the (much stronger) standard security notions

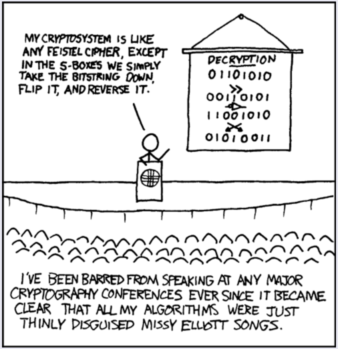
Motivation III

So, what would be great?

Ideally we provide a compiler that

- takes **any** two-party key-exchange protocol and
- **any** authentication protocol
- "blends" them into an AKE
- in a well-established security model

- without knowing the internal mechanisms and
- without modifying the standardized protocols



The standard model by BR

The model introduced by Bellare and Rogaway (CRYPTO '93) is widely adapted.

Execution Environment

Adversaries have the following capabilites (queries):

- Send(m,π): Sends a message m to instance π
- Reveal(π): Reveals the session key k of instance π
- Test(π): Returns a key k_b with $b \in_r \{0, 1\}$, k_0 being the "real" session key k and k_1 being chosen uniformly at random
  - Of course, the adversary must not ask a Reveal(π) query before

The standard model by BR

The model introduced by Bellare and Rogaway (CRYPTO '93) is widely adapted.

Execution Environment

Adversaries have the following capabilites (queries):

- Send(m,π): Sends a message m to instance π
- Reveal(π): Reveals the session key k of instance π
- Test(π): Returns a key k_b with $b \in_r \{0, 1\}$, k_0 being the "real" session key k and k_1 being chosen uniformly at random
  - Of course, the adversary must not ask a Reveal(π) query before

The standard model by BR

The model introduced by Bellare and Rogaway (CRYPTO '93) is widely adapted.

Execution Environment

Adversaries have the following capabilites (queries):

- Send(m,π): Sends a message m to instance π
- Reveal(π): Reveals the session key k of instance π
- Test(π): Returns a key k_b with $b \in_r \{0,1\}$, k_0 being the "real" session key k and k_1 being chosen uniformly at random
  - Of course, the adversary must not ask a Reveal(π) query before

The standard model by BR

The model introduced by Bellare and Rogaway (CRYPTO '93) is widely adapted.

Execution Environment

Adversaries have the following capabilites (queries):

- Send(m,π): Sends a message m to instance π
- Reveal(π): Reveals the session key k of instance π
- Test(π): Returns a key k_b with $b \in_r \{0, 1\}$, k_0 being the "real" session key k and k_1 being chosen uniformly at random
  - Of course, the adversary must not ask a Reveal(π) query before

Security Definitions

An AKE protocol is secure if it holds that

1) Security of the A

- No party P_i communicating with party P_j accepts, if the internal communication transcripts on both sides mismatch

2) Security of the KE

- An adversary cannot determine whether the answer to his Test query was k_0 or k_1 (except for some negligible probability)

Security Definitions

An AKE protocol is secure if it holds that

1) Security of the A

- No party P_i communicating with party P_j accepts, if the internal communication transcripts on both sides mismatch

2) Security of the KE

- An adversary cannot determine whether the answer to his Test query was k_0 or k_1 (except for some negligible probability)

Our results - Two Compilers

First Compiler

- Very efficiently transforms **any** KE into a provably secure AKE in the BR model **without** modifying the KE!
- Proof without random oracles

Second Compiler

- Merges **any** two-party KE with **any** authentication protocol into an AKE (with only minimal changes in the authentication part)
- ... this even works for Zero-Knowledge Authentication
- Proof in the random oracle model

Our results - Two Compilers

First Compiler

- Very efficiently transforms **any** KE into a provably secure AKE in the BR model **without** modifying the KE!
- Proof without random oracles

Second Compiler

- Merges **any** two-party KE with **any** authentication protocol into an AKE (with only minimal changes in the authentication part)
- ... this even works for Zero-Knowledge Authentication
- Proof in the random oracle model

Our results - Two Compilers

First Compiler

- Very efficiently transforms **any** KE into a provably secure AKE in the BR model **without** modifying the KE!
- Proof without random oracles

Second Compiler

- Merges **any** two-party KE with **any** authentication protocol into an AKE (with only minimal changes in the authentication part)
- ... this even works for Zero-Knowledge Authentication
- Proof in the random oracle model

Practical Impact

Example: TLS

- Assuming only that TLS is a passively secure KE (and several results suggest this [MSW08,GMPSS08]) we can construct a provably secure AKE!
- No need to modify the TLS implementation!

An alternative approach would be to provide a proof for full TLS, which is hard in the standard model

Practical Impact

Example: TLS

- Assuming only that TLS is a passively secure KE (and several results suggest this [MSW08,GMPSS08]) we can construct a provably secure AKE!
- No need to modify the TLS implementation!

An alternative approach would be to provide a proof for full TLS, which is hard in the standard model

Practical Impact

Example: TLS

- Assuming only that TLS is a passively secure KE (and several results suggest this [MSW08,GMPSS08]) we can construct a provably secure AKE!
- No need to modify the TLS implementation!

An alternative approach would be to provide a proof for full TLS, which is hard in the standard model

Some related results

What has been done before

- CK01 analyzed the security of IPSEC IKE, but as their result is restricted to only a single protocol it is not comparable to our modular compiler
- BCK98 introduced a modular way to construct authentication and key exchange protocols
- The KY03 compiler adds a signature to every message of a GKE to construct an AKE, but interferes with the KE protocol

Some related results

What has been done before

- CK01 analyzed the security of IPSEC IKE, but as their result is restricted to only a single protocol it is not comparable to our modular compiler
- BCK98 introduced a modular way to construct authentication and key exchange protocols
- The KY03 compiler adds a signature to every message of a GKE to construct an AKE, but interferes with the KE protocol

Some related results

What has been done before

- CK01 analyzed the security of IPSEC IKE, but as their result is restricted to only a single protocol it is not comparable to our modular compiler
- BCK98 introduced a modular way to construct authentication and key exchange protocols
- The KY03 compiler adds a signature to every message of a GKE to construct an AKE, but interferes with the KE protocol

AKE compiler

ssk_A	**A**		**B**	ssk_B
obtain k, T_{KE} from KE		KE ⟷		obtain k, T_{KE} from KE
derive $K = \text{PRF}(k, \text{“}KE\text{“})$ derive $K_{\text{mac}} = \text{PRF}(k, \text{“}MAC\text{“})$				derive $K = \text{PRF}(k, \text{“}KE\text{“})$ derive $K_{\text{mac}} = \text{PRF}(k, \text{“}MAC\text{“})$
	T_1	r_A → ← r_B	T_1	
$\sigma_A := \text{SIG.Sign}(ssk_A, T_{\text{KE}} \| T_1)$				$\sigma_B := \text{SIG.Sign}(ssk_B, T_{\text{KE}} \| T_1)$
	T_2	σ_A → ← σ_B	T_2	
$w_A = \text{MAC}(K_{\text{mac}}, T_2 \parallel '0')$				$w_B = \text{MAC}(K_{\text{mac}}, T_2 \parallel '1')$
	T_3	w_A → ← w_B	T_3	
accept, if $\text{SIG.Vfy}(pk_B, T_1, \sigma_B) = 1$ and $w_B = \text{MAC}(K_{\text{mac}}, T_2 \parallel '1')$				accept, if $\text{SIG.Vfy}(pk_A, T_1, \sigma_A) = 1$ and $w_A = \text{MAC}(K_{\text{mac}}, T_2 \parallel '0')$

Key-Exchange

ssk_A

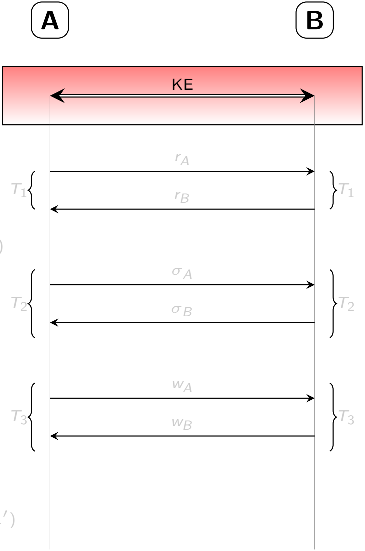
obtain k, T_{KE} from KE

derive $K = \text{PRF}(k, \text{"}KE\text{"})$
derive $K_{mac} = \text{PRF}(k, \text{"}MAC\text{"})$

$\sigma_A := \text{SIG.Sign}(ssk_A, T_{KE} || T_1)$

$w_A = \text{MAC}(K_{mac}, T_2 \parallel {'0'})$

accept, if
$\text{SIG.Vfy}(pk_B, T_1, \sigma_B) = 1$
and $w_B = \text{MAC}(K_{mac}, T_2 \parallel {'1'})$



ssk_B

obtain k, T_{KE} from KE

derive $K = \text{PRF}(k, \text{"}KE\text{"})$
derive $K_{mac} = \text{PRF}(k, \text{"}MAC\text{"})$

$\sigma_B := \text{SIG.Sign}(ssk_B, T_{KE} || T_1)$

$w_B = \text{MAC}(K_{mac}, T_2 \parallel {'1'})$

accept, if
$\text{SIG.Vfy}(pk_A, T_1, \sigma_A) = 1$
and $w_A = \text{MAC}(K_{mac}, T_2 \parallel {'0'})$

KE

- No need to modify the KE protocol
- We only need the transcript and the resulting key

- The KE key k is not used "directly" to enable a standard BR proof
- We derive two keys K and K_{mac} for later use, K being the session key of the resulting AKE

KE

- No need to modify the KE protocol
- We only need the transcript and the resulting key
- The KE key k is not used "directly" to enable a standard BR proof
- We derive two keys K and K_{mac} for later use, K being the session key of the resulting AKE

KE

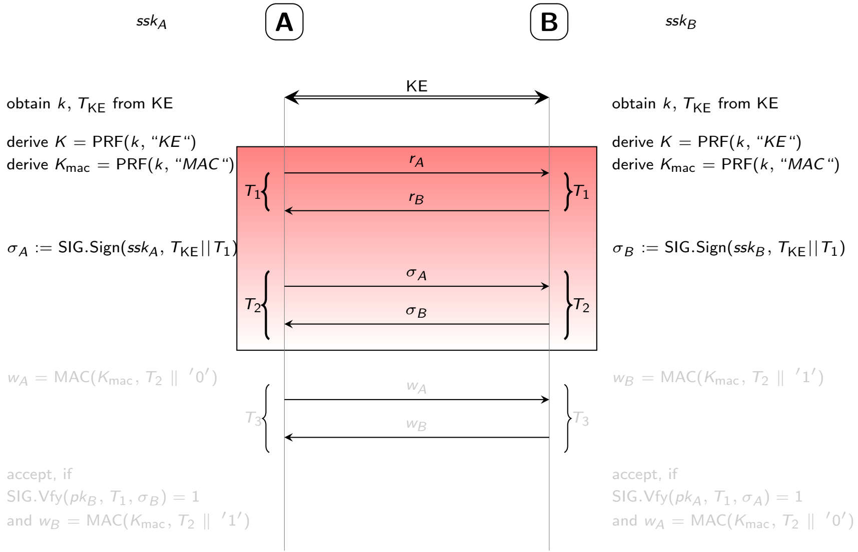
- No need to modify the KE protocol
- We only need the transcript and the resulting key

- The KE key k is not used "directly" to enable a standard BR proof
- We derive two keys K and K_{mac} for later use, K being the session key of the resulting AKE

KE

- No need to modify the KE protocol
- We only need the transcript and the resulting key

- The KE key k is not used “directly” to enable a standard BR proof
- We derive two keys K and K_{mac} for later use, K being the session key of the resulting AKE


ssk_A
A
B
ssk_B
KE
obtain k, T_{KE} from KE
derive $K = \text{PRF}(k, \text{“}KE\text{“})$
derive $K_{mac} = \text{PRF}(k, \text{“}MAC\text{“})$
r_A
r_B
T_1
$\sigma_A := \text{SIG.Sign}(ssk_A, T_{KE}||T_1)$
$\sigma_B := \text{SIG.Sign}(ssk_B, T_{KE}||T_1)$
σ_A
σ_B
T_2
$w_A = \text{MAC}(K_{mac}, T_2 \parallel '0')$
$w_B = \text{MAC}(K_{mac}, T_2 \parallel '1')$
w_A
w_B
T_3
accept, if
$\text{SIG.Vfy}(pk_B, T_1, \sigma_B) = 1$
and $w_B = \text{MAC}(K_{mac}, T_2 \parallel '1')$
accept, if
$\text{SIG.Vfy}(pk_A, T_1, \sigma_A) = 1$
and $w_A = \text{MAC}(K_{mac}, T_2 \parallel '0')$

Signatures

For the authentication part we stick to standard signatures

- Two nonces guarantee freshness of our AKE session
- The long-term secret is used for authentication
- The entire transcript so far is signed to thwart active attacks

Signatures

For the authentication part we stick to standard signatures

- Two nonces guarantee freshness of our AKE session
- The long-term secret is used for authentication
- The entire transcript so far is signed to thwart active attacks

Signatures

For the authentication part we stick to standard signatures

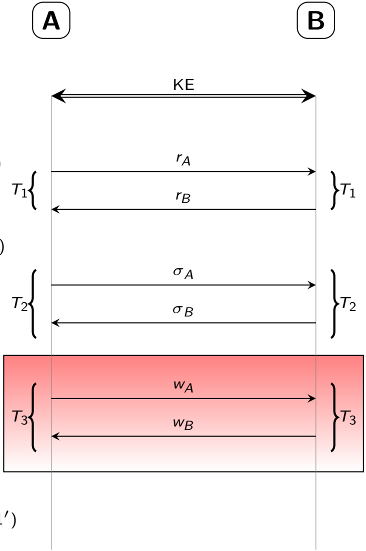
- Two nonces guarantee freshness of our AKE session
- The long-term secret is used for authentication
- The entire transcript so far is signed to thwart active attacks

Signatures

For the authentication part we stick to standard signatures

- Two nonces guarantee freshness of our AKE session
- The long-term secret is used for authentication
- The entire transcript so far is signed to thwart active attacks

ssk_A



ssk_B

Party A:

obtain k, T_{KE} from KE

derive $K = \text{PRF}(k, \text{“}KE\text{“})$
derive $K_{mac} = \text{PRF}(k, \text{“}MAC\text{“})$

$\sigma_A := \text{SIG.Sign}(ssk_A, T_{KE} || T_1)$

$w_A = \text{MAC}(K_{mac}, T_2 \parallel '0')$

accept, if
$\text{SIG.Vfy}(pk_B, T_1, \sigma_B) = 1$
and $w_B = \text{MAC}(K_{mac}, T_2 \parallel '1')$

Party B:

obtain k, T_{KE} from KE

derive $K = \text{PRF}(k, \text{“}KE\text{“})$
derive $K_{mac} = \text{PRF}(k, \text{“}MAC\text{“})$

$\sigma_B := \text{SIG.Sign}(ssk_B, T_{KE} || T_1)$

$w_B = \text{MAC}(K_{mac}, T_2 \parallel '1')$

accept, if
$\text{SIG.Vfy}(pk_A, T_1, \sigma_A) = 1$
and $w_A = \text{MAC}(K_{mac}, T_2 \parallel '0')$

MAC

Including a MAC using K_{mac} at this point serves two purposes

- We enable key confirmation (and "disable" unknown key share attacks)
- We preserve indistinguishability of K due to a "forking trick"

MAC

Including a MAC using K_{mac} at this point serves two purposes

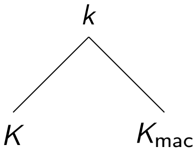
- We enable key confirmation (and “disable” unknown key share attacks)
- We preserve indistinguishability of K due to a “forking trick”

MAC

Including a MAC using K_{mac} at this point serves two purposes

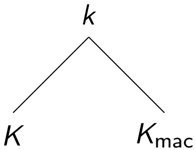
- We enable key confirmation (and "disable" unknown key share attacks)
- We preserve indistinguishability of K due to a "forking trick"

The forking trick



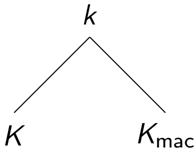
- Revealing the session key k and using it afterwards for the MAC computation enables an adversary to answer the Test query!
- Splitting the session key into two "new" keys enables countering these attacks:
  - Reveal(π) outputs K, but for the MAC computation we use K_{mac}

The forking trick



- Revealing the session key k and using it afterwards for the MAC computation enables an adversary to answer the Test query!
- Splitting the session key into two "new" keys enables countering these attacks:
  - Reveal(π) outputs K, but for the MAC computation we use K_{mac}

The forking trick



- Revealing the session key k and using it afterwards for the MAC computation enables an adversary to answer the Test query!
- Splitting the session key into two "new" keys enables countering these attacks:
  - Reveal(π) outputs K, but for the MAC computation we use K_{mac}

The road so far

So far we presented a compiler with the following properties:

Reminder

- No changes to the KE part
- We excluded passive and active adversaries and even UKS
- ... and the proof is in the BR standard model

Coming up next

- The next compiler provably combines a given KE and a given arbitrary authentication protocol *in the random oracle model*

The road so far

So far we presented a compiler with the following properties:

Reminder

- No changes to the KE part
- We excluded passive and active adversaries and even UKS
- … and the proof is in the BR standard model

Coming up next

- The next compiler provably combines a given KE and a given arbitrary authentication protocol *in the random oracle model*

The road so far

So far we presented a compiler with the following properties:

Reminder

- No changes to the KE part
- We excluded passive and active adversaries and even UKS
- ... and the proof is in the BR standard model

Coming up next

- The next compiler provably combines a given KE and a given arbitrary authentication protocol *in the random oracle model*

The road so far

So far we presented a compiler with the following properties:

Reminder

- No changes to the KE part
- We excluded passive and active adversaries and even UKS
- ... and the proof is in the BR standard model

Coming up next

- The next compiler provably combines a given KE and a given arbitrary authentication protocol *in the random oracle model*

The road so far

So far we presented a compiler with the following properties:

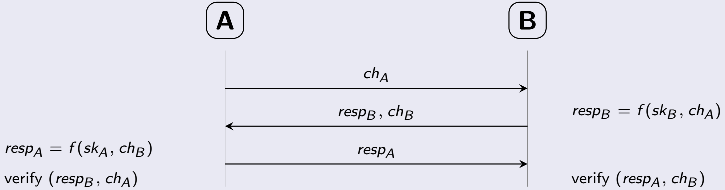
Reminder

- No changes to the KE part
- We excluded passive and active adversaries and even UKS
- ... and the proof is in the BR standard model

Coming up next

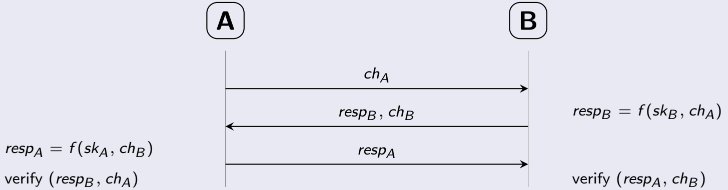
- The next compiler provably combines a given KE and a given arbitrary authentication protocol *in the random oracle model*

Standard Challenge-Response protocol

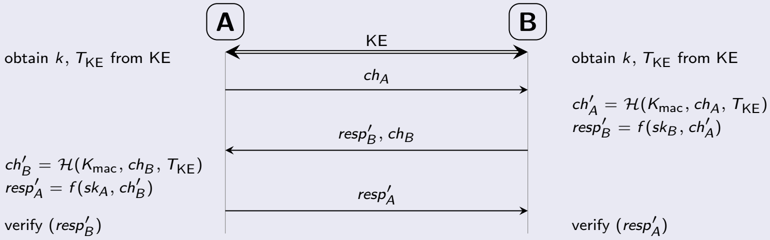


Our compiler

Standard Challenge-Response protocol



Our compiler



Structure

The ideas are quite similar as compared to the first compiler:

- Again we take the transcript from the KE
    - Remark: We still need the forking trick to proof security
- ... but this time we can use (nearly) **any** authentication protocol secure against active attacks

Structure

The ideas are quite similar as compared to the first compiler:

- Again we take the transcript from the KE
  - Remark: We still need the forking trick to proof security
- ... but this time we can use (nearly) **any** authentication protocol secure against active attacks

Structure

The ideas are quite similar as compared to the first compiler:

- Again we take the transcript from the KE
  - Remark: We still need the forking trick to proof security
- ... but this time we can use (nearly) **any** authentication protocol secure against active attacks

Summary

We presented two compilers with the following properties:

First Compiler

- No changes to the KE part
- We excluded passive and active adversaries and even UKS
- ... and the proof is in the BR standard model

Second Compiler

- Again no changes to the KE part
- We can use **any** authentication protocol with only minimal changes
- ... but our proof makes use of a random oracle

Summary

We presented two compilers with the following properties:

First Compiler

- No changes to the KE part
- We excluded passive and active adversaries and even UKS
- ... and the proof is in the BR standard model

Second Compiler

- Again no changes to the KE part
- We can use **any** authentication protocol with only minimal changes
- ... but our proof makes use of a random oracle

Summary

We presented two compilers with the following properties:

First Compiler

- No changes to the KE part
- We excluded passive and active adversaries and even UKS
- ... and the proof is in the BR standard model

Second Compiler

- Again no changes to the KE part
- We can use **any** authentication protocol with only minimal changes
- ... but our proof makes use of a random oracle

Summary

We presented two compilers with the following properties:

First Compiler

- No changes to the KE part
- We excluded passive and active adversaries and even UKS
- ... and the proof is in the BR standard model

Second Compiler

- Again no changes to the KE part
- We can use **any** authentication protocol with only minimal changes
- ... but our proof makes use of a random oracle

Summary

We presented two compilers with the following properties:

First Compiler

- No changes to the KE part
- We excluded passive and active adversaries and even UKS
- ... and the proof is in the BR standard model

Second Compiler

- Again no changes to the KE part
- We can use **any** authentication protocol with only minimal changes
- ... but our proof makes use of a random oracle

Summary

We presented two compilers with the following properties:

First Compiler

- No changes to the KE part
- We excluded passive and active adversaries and even UKS
- ... and the proof is in the BR standard model

Second Compiler

- Again no changes to the KE part
- We can use **any** authentication protocol with only minimal changes
- ... but our proof makes use of a random oracle

Summary

We presented two compilers with the following properties:

First Compiler

- No changes to the KE part
- We excluded passive and active adversaries and even UKS
- ... and the proof is in the BR standard model

Second Compiler

- Again no changes to the KE part
- We can use **any** authentication protocol with only minimal changes
- ... but our proof makes use of a random oracle

Future work

- Try to find more efficient (specialized) variants of our compilers
- Implement our compiler for real-world protocols
- Extend our results to group key exchange protocols

Any questions??

Future work

- Try to find more efficient (specialized) variants of our compilers
- Implement our compiler for real-world protocols
- Extend our results to group key exchange protocols

Any questions??

Future work

- Try to find more efficient (specialized) variants of our compilers
- Implement our compiler for real-world protocols
- Extend our results to group key exchange protocols

Any questions??

Future work

- Try to find more efficient (specialized) variants of our compilers
- Implement our compiler for real-world protocols
- Extend our results to group key exchange protocols

Any questions??

Proof part 1

Session freshness

We show the session freshness by applying the birthday bound

Matching Conversation I

We exclude active adversaries against T_{KE}, T_1 and T_2 by the EUF-CMA security of the digital signature scheme $\Rightarrow$ the adversary is restricted to passive attacks against the KE

Key indistinguishability of k

We show key indistinguishability of k by the (passive) security of the KE

Proof part 2

Key indistinguishability of K and K_{mac}

We show key indistinguishability by the security of the PRF

Matching Conversation II

We exclude active adversaries against T_3 by the security of the MAC