Practical and Tightly-Secure Digital Signatures and Authenticated Key Exchange

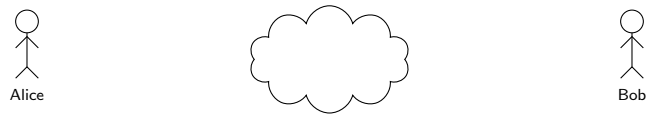
Kristian Gjøsteen[1] Tibor Jager[2]

NTNU - Norwegian University of Science and Technology, Trondheim, Norway,
kristian.gjosteen@ntnu.no

Paderborn University, Paderborn, Germany, tibor.jager@upb.de

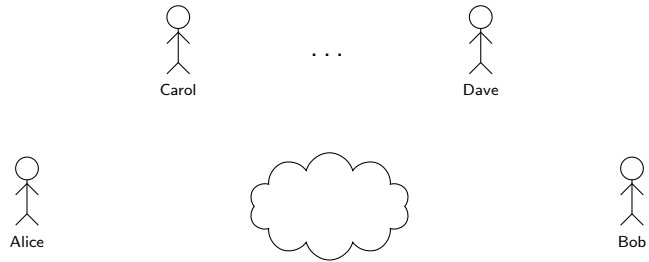
CRYPTO 2018

Authenticated Key Exchange



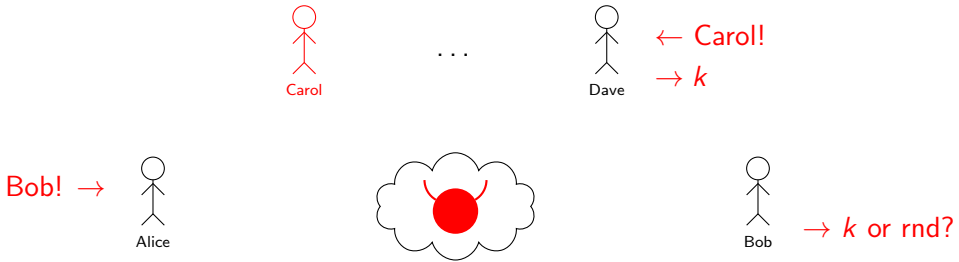
Alice and Bob want to exchange a key.

Authenticated Key Exchange



Alice and Bob and Carol and . . . want to exchange keys.

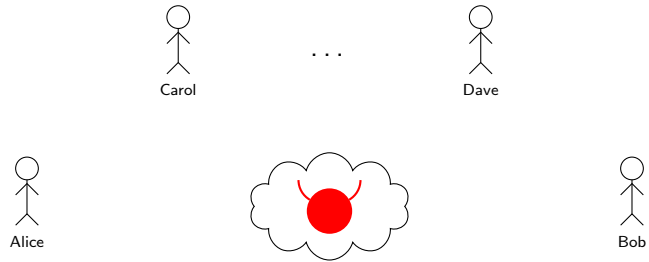
Authenticated Key Exchange



The adversary

- directs honest users' key exchanges;
- can inspect keys or test keys (real-or-random);
- controls the network;
- and can adaptively corrupt users.

Authenticated Key Exchange



Our goal: If

- Alice believes she has exchanged a key with Bob,

it must be true that

- Bob exchanged exactly the same key exactly once with Alice; and
- it looks random.

Why Security Proofs?

A typical security proof is a reduction from some problem to attacking our crypto:

> *A poly-time crypto adversary with non-negligible advantage*
> *gives us*
> *a poly-time problem solver with non-negligible advantage.*

The scheme is secure if we believe no such solver exists.

Why do we want them?

- Why not?
- Ensure that our system is not trivially breakable.
- Help us choose security parameters.

Why Security Proofs?

A typical security proof is a reduction from some problem to attacking our crypto:

> *A crypto adversary using time T_1 with advantage ϵ_1*
> *gives us*
> *a problem solver using time T_2 with advantage ϵ_2.*

The scheme is secure if we believe no such solver exists.

Why do we want them?

- Why not?
- Ensure that our system is not trivially breakable.
- Help us choose security parameters.

Why Tight Security Proofs?

A typical security proof is a reduction from some problem to attacking our crypto:

> *A crypto adversary using time T_1 with advantage ϵ_1*
> *gives us*
> *a problem solver using time T_2 with advantage ϵ_2.*

The scheme is secure if we believe no such solver exists.

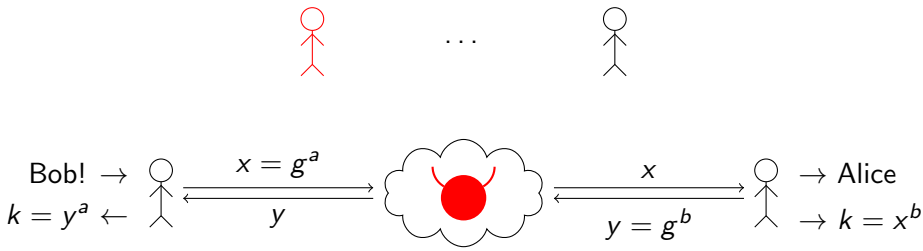
- A reduction is tight if $T_1 \approx T_2$ and $\epsilon_1 \approx \epsilon_2$.

Why do we want them?

- Why not?
- Ensure that our system is not trivially breakable.
- Help us choose security parameters.

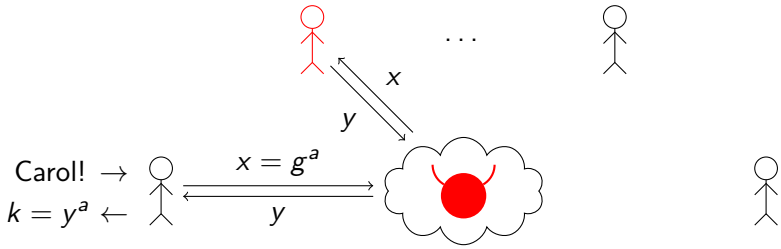
Plain Diffie-Hellman

No Tampering + Static Corruption = No Problem



Plain Diffie-Hellman

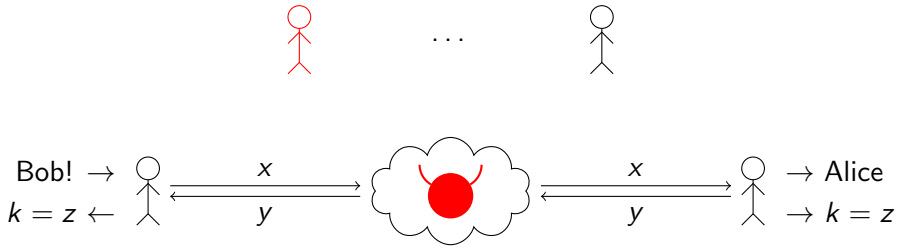
No Tampering + Static Corruption = No Problem



When Alice talks to a corrupted Carol, we run Diffie-Hellman as usual.

Plain Diffie-Hellman

No Tampering + Static Corruption = No Problem

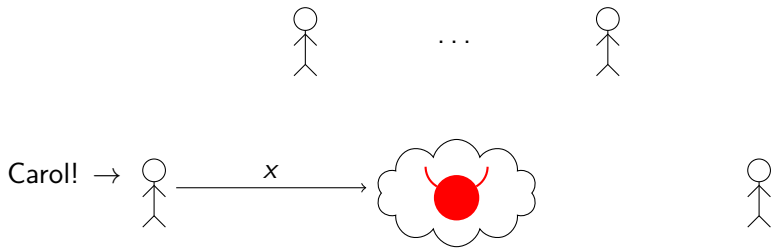


When Alice talks to an honest Bob, we simulate the conversation using a Diffie-Hellman tuple (g, x, y, z).

Rerandomization gives us many tuples and a tight proof.

Plain Diffie-Hellman

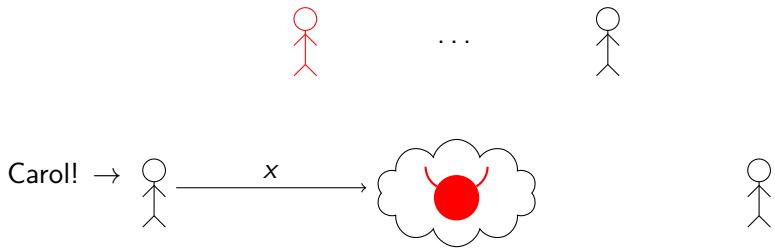
No Tampering + Adaptive Corruption = Commitment Problem



The commitment problem: the adversary may corrupt the responder after we have committed by sending the first message.

Plain Diffie-Hellman

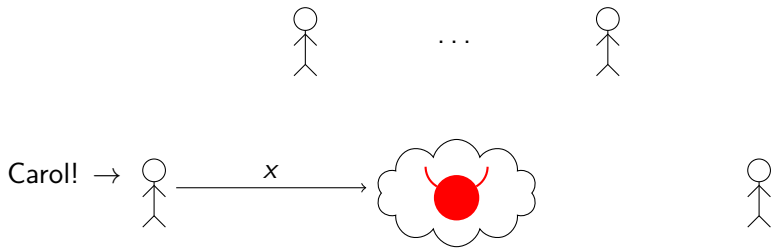
No Tampering + Adaptive Corruption = Commitment Problem



The commitment problem: the adversary may corrupt the responder after we have committed by sending the first message.

Plain Diffie-Hellman

No Tampering + Adaptive Corruption = Commitment Problem

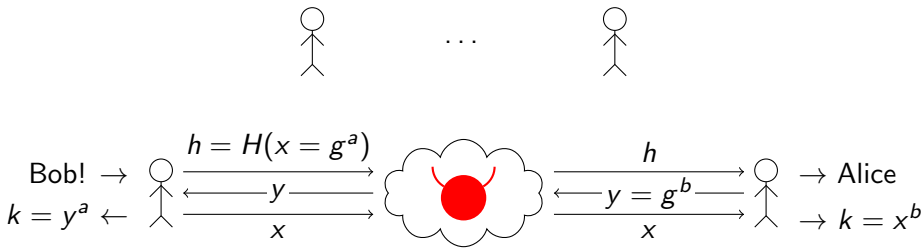


The commitment problem: the adversary may corrupt the responder after we have committed by sending the first message.

We can guess a communicating pair, but then tightness is lost.

Our Modified Diffie-Hellman

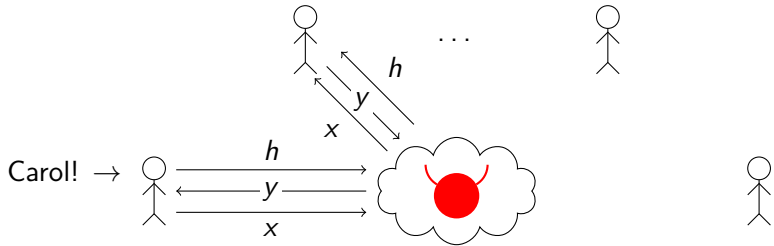
No Tampering + Adaptive Corruption = No Problem



- We hash the first message, and send x only after receiving the response.

Our Modified Diffie-Hellman

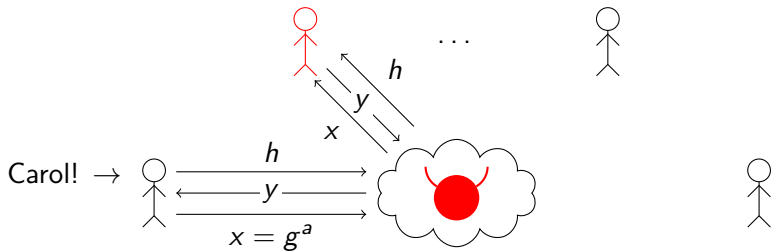
No Tampering + Adaptive Corruption = No Problem



- We hash the first message, and send x only after receiving the response.
- In the random oracle model, our reduction does not have to commit to x until we know if the response was honest or not. We get a tight reduction.

Our Modified Diffie-Hellman

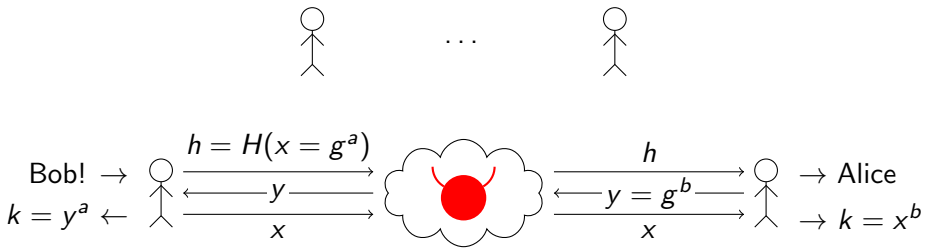
No Tampering + Adaptive Corruption = No Problem



- We hash the first message, and send x only after receiving the response.
- In the random oracle model, our reduction does not have to commit to x until we know if the response was honest or not. We get a tight reduction.

Our Modified Diffie-Hellman

No Tampering + Adaptive Corruption = No Problem



- We hash the first message, and send x only after receiving the response.
- Note that there is an additional message flow compared to plain Diffie-Hellman. The responder also learns the key later. This is often not a problem.

Security Parameters and Performance

We compare our protocol with plain Diffie-Hellman.

We want 128-bit security. Our protocol will use the NIST P-256 curve.

Security Parameters and Performance

We compare our protocol with plain Diffie-Hellman.

We want 128-bit security. Our protocol will use the NIST P-256 curve.

For plain Diffie-Hellman:

- Small-scale: 2^{16} users, 2^{16} sessions and quadratic loss 2^{64}: NIST P-384.
- Large-scale: 2^{32} users, 2^{32} sessions and quadratic loss 2^{128}: NIST P-521.

Exponentiation relative time cost: P-256 $= 1$, P-384 $= 2.7$, P-521 $= 7.7$.

Security Parameters and Performance

We compare our protocol with signed Diffie-Hellman.

We want 128-bit security. Our protocol will use the NIST P-256 curve.

For signed Diffie-Hellman:

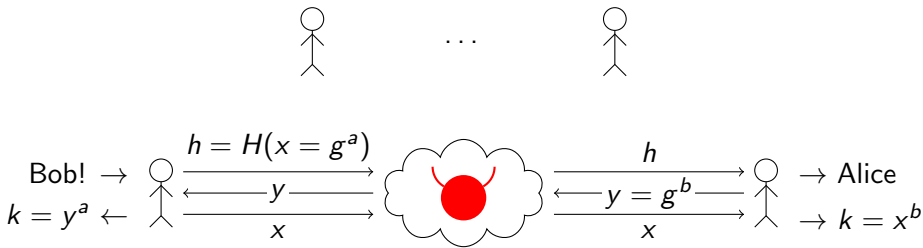
- Small-scale: 2^{16} users, 2^{16} sessions and quadratic loss 2^{64}: NIST P-384.
- Large-scale: 2^{32} users, 2^{32} sessions and quadratic loss 2^{128}: NIST P-521.

Exponentiation relative time cost: P-256 $= 1$, P-384 $= 2.7$, P-521 $= 7.7$.

		small-scale		large-scale	
	# exps.	# bits	time	# bits	time
Our scheme	2	770	2	770	2
Plain DH	2	770	5.4	1044	15.4

Our Modified Diffie-Hellman

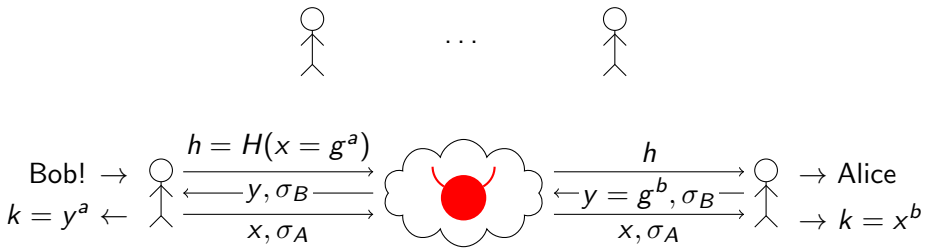
~~No~~ Tampering + Adaptive Corruption = A Need for Signatures



- If the adversary is allowed to tamper with messages, this protocol obviously fails.

Our Modified Diffie-Hellman

~~No~~ Tampering + Adaptive Corruption = A Need for Signatures



- If the adversary is allowed to tamper with messages, this protocol obviously fails.
- The natural answer is to add the usual signatures.
- But this requires a signature scheme with a tight reduction!

Signatures with Tight Reductions

The standard security notion for signatures considers only a single key pair.

- There is a standard reduction to many key pairs, but it is non-tight.
- In fact, the loss in the standard reduction is quadratic in the number of users, because of adaptive compromise.

Signatures with Tight Reductions

The standard security notion for signatures considers only a single key pair.

- There is a standard reduction to many key pairs, but it is non-tight.
- In fact, the loss in the standard reduction is quadratic in the number of users, because of adaptive compromise.

The main obstacle to a tight reduction:

- We need to know every secret key to respond to compromise.
- We need to extract something from the eventual forgery.

Signatures with Tight Reductions

The standard security notion for signatures considers only a single key pair.

- There is a standard reduction to many key pairs, but it is non-tight.
- In fact, the loss in the standard reduction is quadratic in the number of users, because of adaptive compromise.

The main obstacle to a tight reduction:

- We need to know every secret key to respond to compromise.
- We need to extract something from the eventual forgery.

Tight reductions are impossible for schemes with unique signatures or signing keys.

“Double-signature” idea

The “double-signature” idea from Bader et al. (TCC 15).

- The user has “real” and a “fake” verification key.
- A signature consists of a real signature for the “real” verification key, and a fake signature for the “fake” verification key.
- We embed our hard problem in the “fake” verification key.
- If “real” and “fake” keys and signatures are indistinguishable, the adversary will produce a forgery that applies to the “fake” verification key with probability $1/2$.

This “double-signature” idea does not work for most signature schemes. The only previous construction is impractical.

Our Signature Scheme

Our basis is the The Goh-Jarecki signature scheme, which uses a cyclic group G and a hash function $H : \{0,1\}^* \to G$.

Our Signature Scheme

Our basis is the The Goh-Jarecki signature scheme, which uses a cyclic group G and a hash function $H : \{0,1\}^* \to G$.

Verification key: $y = g^a$.

Signature: $z = H(m)^a$ and a proof that $\log_{H(m)} z = \log_g y$.

Our Signature Scheme

Our basis is the The Goh-Jarecki signature scheme, which uses a cyclic group G and a hash function $H : \{0,1\}^* \to G$.

Verification key: $y = g^a$.

Signature: $z = H(m)^a$ and a proof that $\log_{H(m)} z = \log_g y$.

Our construction:

- We combine two signatures using an OR-proof.
- The fake signature is just a random z and a simulated proof of equal discrete logarithms.

Our Signature Scheme

Our basis is the The Goh-Jarecki signature scheme, which uses a cyclic group G and a hash function $H : \{0,1\}^* \to G$.

Verification key: $y = g^a$.

Signature: $z = H(m)^a$ and a proof that $\log_{H(m)} z = \log_g y$.

Our construction:

- We combine two signatures using an OR-proof.
- The fake signature is just a random z and a simulated proof of equal discrete logarithms.

The usual AKE security models require strongly unforgeable signatures. This scheme is not strongly unforgeable, so we need to use a slightly different security model.

Security Parameters and Performance

We compare our protocol with Diffie-Hellman.

We want 128-bit security. Our protocol will use the NIST P-256 curve.

For Diffie-Hellman:

- Small-scale: 2^{16} users, 2^{16} sessions and quadratic loss 2^{64}: NIST P-384.
- Large-scale: 2^{32} users, 2^{32} sessions and quadratic loss 2^{128}: NIST P-521.

Exponentiation relative time cost: P-256 = 1, P-384 = 2.7, P-521 = 7.7.

		small-scale		large-scale	
	# exps.	# bits	time	# bits	time
Our scheme	2	770	2	770	2
Plain DH	2	770	5.4	1044	15.4
Our scheme II	17	4358	17	4358	17
DH+ECDSA	5	2306	13.5	3128	38.5

Summary

- First practical tightly-secure signature scheme with adaptive corruptions.
- First practical tightly-secure AKE.
- More efficient than ordinary signed Diffie-Hellman for large-scale deployment settings

Questions?