

# Exchanging a key - *how hard can it be?*

**Cas Cremers**

Joint work with Michèle Feltz

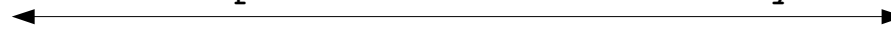


# Authenticated Key Exchange Protocols

$(a, g^a)$



use AKE protocol to establish key  $k$



„Hi, do you want to visit us? I'll make sure the university pays for the beers.“



„Great, but only if Cas is not coming“



$(b, g^b)$



## ■ Perfect Forward Secrecy

- If all long-term keys are compromised, then old sessions still secure
  - Easy to achieve?

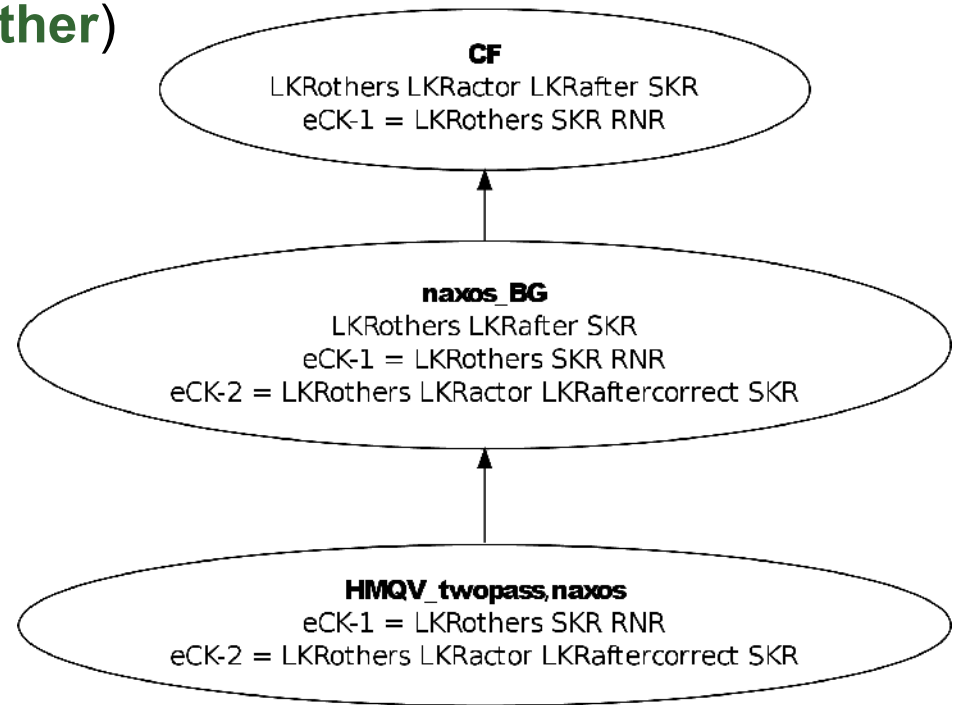
## ■ Deniability

- A party should be able to deny having sent a particular message
  - Some tension with authentication

# Exchanging a key – how hard can it be?

- My background: **Formal Analysis of Security Protocols**

- Symbolic models & tools (**Scyther**)
- Context: narrowing the gap between symbolic and computational analysis



- **Roadmap:**

- Current security models and properties: PFS, deniability
- Our proposals: **eCK-PFS** and **peer-and-time deniability**
- New protocol

# Protocol example: (H)MQV

$$\begin{aligned} A \rightarrow B : & \quad g^x && \text{(Check that } g^x \in G) \\ B \rightarrow A : & \quad g^y && \text{(Check that } g^y \in G) \end{aligned}$$

Session key:

$$H(((g^y)(g^b)^e)^{x+da}) = H(g^{(x+da)(y+eb)}) = H(((g^x)(g^a)^d)^{y+eb})$$

(Long-term private key of  $A$  is  $a$ , public key is  $g^a$ .)

	d	e
MQV	$2^l + (g^x \bmod 2^l)$	$2^l + (g^y \bmod 2^l)$
HMQV	$H'(g^x, B)$	$H'(g^y, A)$

# Security properties of AKE protocols

- Main desired security property:
  - **Adversary cannot distinguish established key from random**
- However: **Complex system!**
  - multiple network messages,
  - stateful programs,
  - long-term/short term keys,
  - intermediate computations,
  - behaviour over longer time periods...
- **What exactly can the adversary do?**
  - Motivation for range of models and corresponding protocols

# Bellare Rogaway 1993: Back to the roots

- Model **interaction between the adversary (a PPT) and the agents** performing the protocol
- Adversary **completely controls network** and **can learn some session keys**
  - “**send query**” : start local session, send a message to a local session and get the response message
  - “**reveal query**” : reveal the session key computed in a local session

# BR93: security notion

- Security defined in terms of a game:
  - Adversary may do **send** and **reveal** queries
  - Then he can choose one local session to attack, usually called “**Test**”
    - A coin ***b*** is flipped:
      - If ***b*** = 0, Adversary receives the session key of the **Test** session
      - If ***b*** = 1, Adversary receives a random bit string (from the key space)
  - Do some more queries
  - Now adversary may guess what ***b*** was
- If guessed bit equal to ***b***, **adversary wins**
- Security notion is satisfied if adversary has only a negligible advantage over simply guessing ***b***

# BR93

- However, the adversary can always win the previous game
  - Something is wrong!
- **Problem 1:** If adversary can reveal the session key of the `Test` session (using the `reveal(Test)` query)...
  - **Solution:** Disallow `reveal(Test)`
- **Problem 2:** computed key is *supposed* to be shared with some other session – what if we reveal that one?
  - **Solution:** Define partner by **matching histories**
    - *Partner* = sent and received messages identically with `Test`
    - Disallow `reveal(Test)` and `reveal(Partner)`

# Partnering: trouble then, trouble now



- Original intent: which **complete sessions** must compute **the same key**?
  - Matching histories
  - Pre-shared session IDs
  - Matching histories + identity information
  - Maybe matching initiators should compute the same key as well... add variants to the above
- Later use: which **incomplete sessions** store **related randomness/state**?
  - Possibly bad model of real attacks
    - At time  $t$ , compromise some local sessions of  $A$  but not all?
    - Models side-channel attacks, but not much else

# More attacks, more advanced properties

- Assume:  $A$  performs the Test session and tries to communicate with  $B$
- What about learning Long-term keys?
  - Allow **Corrupt/Ltk-reveal** of  $C, D, \dots$
  - **Key Compromise Impersonation (KCI)**
    - Allow **Ltk-reveal** of  $A$
    - Protocol can still work – we only need to
  - **Perfect Forward Secrecy (PFS)**
    - Allow **Ltk-reveal** of  $A$  and  $B$  – but only after the end of the **Test** session

# Well, maybe not PFS in two messages...

1.  $\rightarrow B: g^x$  ( $B$  Checks that  $g^x \in G$ )
2.  $B \rightarrow : g^y$

- Generic attack (Krawczyk):
  - 1. Adversary chooses arbitrary  $x$  and sends  $g^x$  to  $B$  (as  $A$ )
  - 2.  $B$  accepts, sends response, computes session key  $k$
  - 3. Adversary chooses  $B$ 's session as the **Test** session
  - 4.  $B$  ends his session (or the key expires)
  - 5. **Ltk-reveal** of  $A$
  - 6. Adversary can recompute any key that an honest  $A$  could have computed on the basis of  $x$  and  $A$ 's long-term key.
- Motivated the introduction of weak-PFS

# Example model: extended-CK (eCK)

- Queries
  - **Send** send a message to, or initiate, a local session
  - **Sk-reveal** reveal the session key computed by a local session
  - **Ltk-reveal** reveal the long-term key of a party
  - **Ephk-reveal** reveal the ephemeral key of a local session (e.g.  $x,y$ )
- An experiment is valid **unless**:
  - (Assume Test performed by  $A$ , supposedly with  $B$ )
  - **Sk-reveal**( $Isid$ ), where  $Isid$  is **Test** or a matching session
  - Both **Ltk-reveal**( $A$ ) and **Ephkey-reveal**(**Test**)
  - If a matching session  $Isid$  exists:
    - Both **Ltk-reveal**( $B$ ) and **Ephkey-reveal**( $Isid$ )
  - If no matching session exists:
    - **Ltk-reveal**( $B$ )

# Security notions versus reality



# Same difference

# Models

Individual features

	CK	CK <sub>HMQV</sub>	eCK
domain restrictions	protocol messages contain session identifier		
behaviour restrictions			adversary passive during communication between test session and its eCK matching session
sessions that should compute the same key	CK-matching ( $\approx_D$ )	CK <sub>HMQV</sub> -matching ( $\approx_A$ )	eCK-matching ( $\approx_C$ )
incompatible key equivalence types for role-symmetric protocols	$\approx_B, \approx_C$	$\approx_B, \approx_C$	$\approx_A, \approx_B$
reveal long-term key of test <sub>A</sub>	if test has expired, test can be corrupted		ephemeral keys of test not revealed
reveal long-term key of test <sub>B</sub>	if test has expired, test can be corrupted	[wPFS]: if session that CK-matches test exists, and both test and the matching session are CK <sub>HMQV</sub> -clean; [basic,KCI,eph]: never	if eCK-matching sessions exist, and the ephemeral keys of these sessions are not revealed
reveal ephemeral keys of s	if ephemeral keys are in session state and $s \neq \text{test}$ and s is not CK-matching test	[basic,KCI,wPFS]: never [eph]: anytime	if $s = \text{test}$ , the long-term key of $s_A$ must not be revealed; if s eCK-matches test, the long-term key of $s_B$ must not be revealed; otherwise allowed
reveal session keys of $s \neq \text{test}$	if s is not CK-matching test	if s is not CK <sub>HMQV</sub> -matching test	if s is not eCK-matching test
reveal other session state of s (e.g. intermediate computations)	if $s \neq \text{test}$ and s is not CK-matching test	if $s \neq \text{test}$ and s is not CK <sub>HMQV</sub> -matching test	never

Incomparable!

From: *Examining Indistinguishability-Based Security Models for Key Exchange Protocols: The case of CK, CK-HMQV, and eCK*. ASIACCS 2011

# Revisiting Krawczyk's attack

- **Can no 2-message AKE protocol satisfy PFS** in the context of an active adversary?
- Why not just use **signatures**?
  - Prevents the adversary from inserting his own messages
- Possible **reasons to avoid authenticating** the messages?
  - Efficiency
  - **Deniability?**

# Deniability

- AKE definition by Di Raimondo, Gennaro, Krawczyk
- Very strong notion inspired by zero-knowledge proofs
  - Alice can deny everything (existence!)
- Seems mostly historical, too strong for most protocols
- Weaker form shown for **sigma** protocols
- **sigma** protocols sign received data...
  - Adversary could insert message digest of today's newspaper and have it signed
    - shows that Alice was willing to communicate today or later

## Can we do better?

- Integrate Perfect Forward Secrecy into the eCK model
- See if we can still have (some form of) deniability in a reasonable protocol

# Security model: eCK-like

- We modify eCK in two ways:
  - Use “**origin session**” instead of “**matching session**” for ephemeral key reveal restrictions
    - Captures “what I received was generated in a real session”  
(more natural for exclusion, and simplifies PFS integration)
  - Allow the adversary to also learn **B**'s long-term keys after the end of the **Test** session
    - Only minimal restriction: not also learning the received ephemeral key (by origin session)

# Proposed model: eCK-PFS

- Queries
  - **Send** send a message to, or initiate, a local session
  - **Sk-reveal** reveal the session key computed by a local session
  - **Ltk-reveal** reveal the long-term key of a party
  - **Ephk-reveal** reveal the ephemeral key of a local session (e.g.  $x,y$ )
- An experiment is valid **unless**:
  - (Assume **Test** performed by **A**, supposedly with **B**)
  - **Sk-reveal**(*Isid*), where *Isid* is **Test** or a matching session
  - Both **Ltk-reveal**(**A**) and **Ephkey-reveal**(**Test**)
  - If a matching **origin** session *Isid* exists:
    - Both **Ltk-reveal**(**B**) and **Ephkey-reveal**(*Isid*)
  - If no matching **origin** session exists:
    - **Ltk-reveal**(**B**) **before the end of Test**

# Related models?

- No stronger model proposed
  - no model with ephemeral key reveal also has PFS integrated
- No known two-message protocols secure in eCK-PFS!
  - Insecure in eCK-PFS:
    - (H)MQV
    - YAK
    - NAXOS with Boyd-Gonzales transformation
    - mOT
- “Strongest adversary” myth:
  - YAK and NAXOS have claims of “strongest possible adversary”
  - First: Depends on choice of modeling elements
  - Second: even given fixed modeling elements, eCK is not the strongest

# Peer-and-Time Deniability

- We need some form of authentication to counter the generic attack, even when **A**'s long-term private key is revealed (KCI) → *signatures*
  - A can no longer deny having signed something
- Peer deniability:
  - A can deny to ever have been willing to talk to B
- Time deniability
  - A can deny having been active during any specific time frame
- In practice: “I was willing to run the protocol years ago with C, but I never completed any session.”

# Protocol

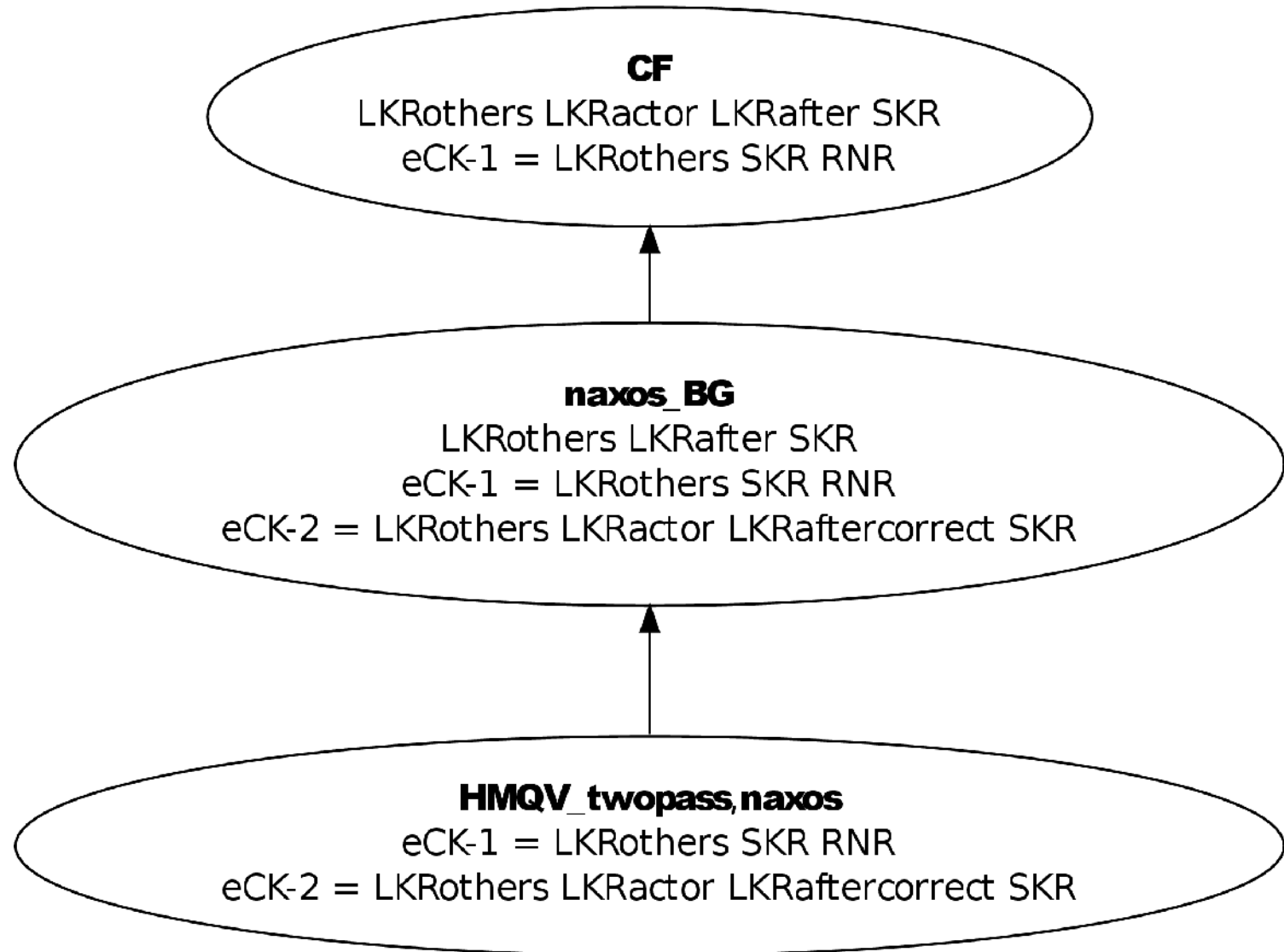
$$\begin{array}{ll} A \rightarrow B : & A, g^x, \sigma_A(g^y) \quad (\text{Check that } g^x \in G) \\ B \rightarrow A : & B, g^y, \sigma_B(g^y) \quad (\text{Check that } g^y \in G) \end{array}$$

Session key:

$$KDF(A, B, (g^y g^b)^{(x+a)}, g^x) = KDF(A, B, g^{(x+a)(y+b)}, g^x) = \dots$$

- MQV-like with signatures of self-generated data
- Dropped d,e because received values are signed
- Group element check still needed

# Back to the graph: auto-generated\* by the Scyther tool (~1h)



(\*) No brains were hurt in the making of this graph

# Conclusions

- New Security model **eCK-PFS** strengthens eCK with PFS
  - Thought to be impossible to satisfy
- New deniability notion: **peer-and-time deniability**
  - Satisfiable by reasonable protocols
  - Provides a useful level of deniability
- New protocol that satisfies eCK-PFS and peer-and-time deniability
  - Maybe time for a different approach to AKE security?
  - Encouraging: these are only side-effects of revisiting the models while working on automation.

