Secure Messaging in Mobile Environments

Sebastian Verschoor

Eindhoven University of Technology

December 7th, 2015

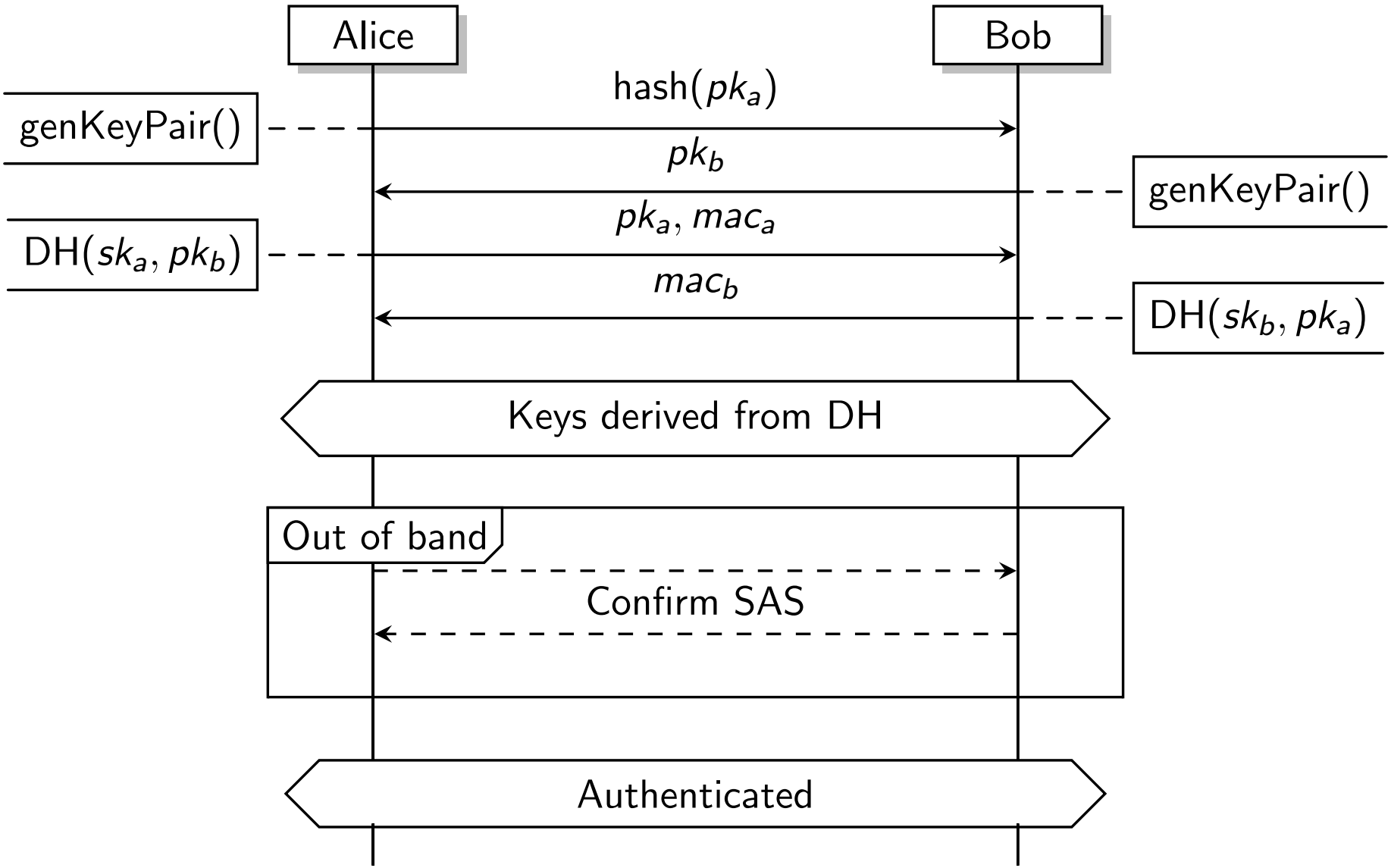
TU/e

Outline

Introduction

- Silent Circle instant messaging protocol (SCimp)
- Built on XMPP
- Mobile environment
  - Asynchronous
- Security properties
  - Confidentiality/Integrity/Availability
  - Authentication of other party
  - Deniability
  - Key erasure
  - Future secrecy
  - (Privacy Protection)

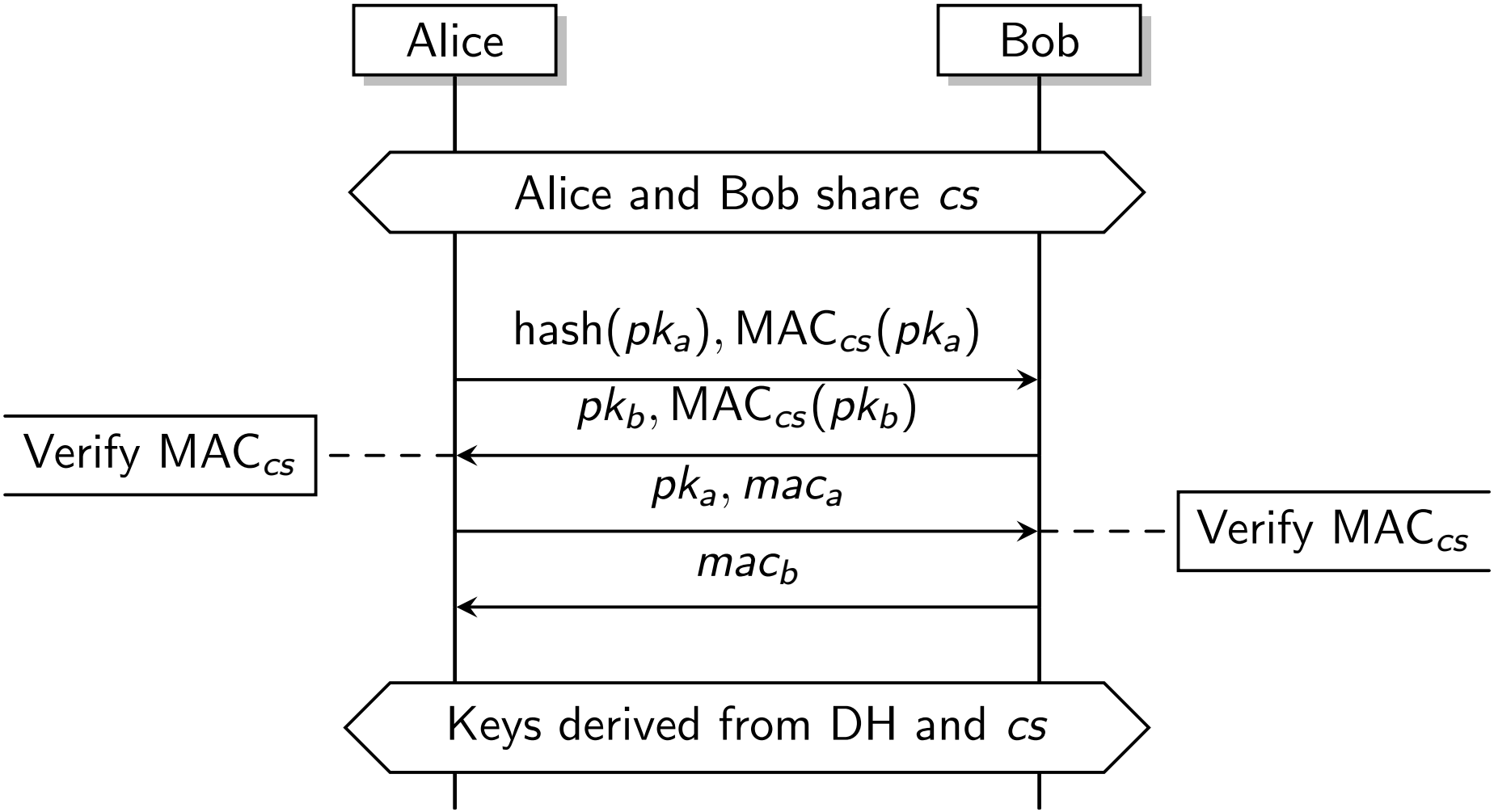
SCimp: Key negotiation



SCimp: Key negotiation

- ECDHE gives shared secret Z, from which are derived:
  - $k_{snd,0}, k_{rcv,0}, i_{snd,0}, i_{rcv,0}$; for message encryption and authentication
  - mac_a, mac_b; to confirm knowledge of Z
  - SAS; for authentication of identity
  - cs; for rekeying
- User messages can be sent after four key exchange messages
- SAS confirms identity all previous communication
  - Requires commitment to pk_a to prevent collision attack
- Completely ephemeral
  - Deniable

SCimp: Rekeying



SCimp: Rekeying

- First: store old decryption key (messages might arrive out of order)
- Optional: SAS comparison only after several rekeyings
- Rekeying ensures *future secrecy*
- It is not specified when to rekey
- Protocol aborts on error
  - Keys are discarded, including *cs*

SCimp: Sending data

- Encrypt
  - ciphertext $= \mathrm{AES}_{k_j}(i_j, \text{plaintext})$

SCimp: Sending data

- Encrypt
  - ciphertext $= \mathrm{AES}_{k_j}(i_j, \text{plaintext})$
- Update keys (ratchet)
  - $k_{j+1} = \mathrm{MAC}_{k_j}(i_j)$
  - $i_{j+1} = i_j + 1$

SCimp: Sending data

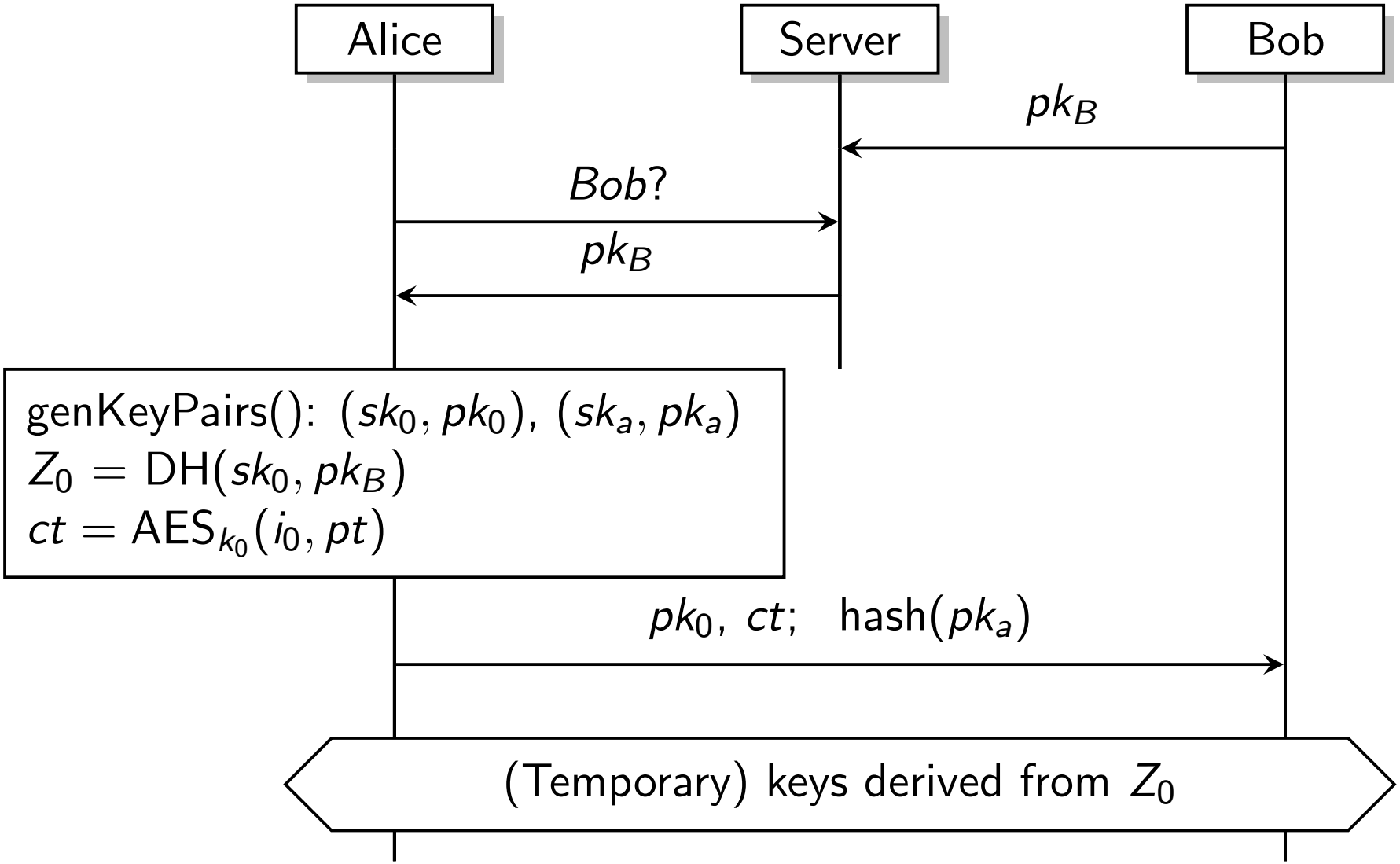
- Encrypt
  - ciphertext $= \mathrm{AES}_{k_j}(i_j, \text{plaintext})$
- Update keys (ratchet)
  - $k_{j+1} = \mathrm{MAC}_{k_j}(i_j)$
  - $i_{j+1} = i_j + 1$
- Send message:
  - i_j
  - ciphertext

SCimp: Sending data

- Encrypt
  - ciphertext $= \mathrm{AES}_{k_j}(i_j, \text{plaintext})$
- Update keys (ratchet)
  - $k_{j+1} = \mathrm{MAC}_{k_j}(i_j)$
  - $i_{j+1} = i_j + 1$
- Send message:
  - i_j
  - ciphertext

- No message signatures: deniable
- Ratchet enables key erasure, but:
  - Out of order messages require you to store old keys
  - Old keys compromise future keys

SCimp v2: Progressive encryption



SCimp v2: Progressive encryption

- SAS confirmation after “regular” key negotiation
  - Confirms entire conversation

SCimp v2: Progressive encryption

- SAS confirmation after “regular” key negotiation
  - Confirms entire conversation
- Vulnerable to Man In The Middle (MITM) attack‘
  - MITM re-encrypts and forwards user messages
  - MITM blocks keying messages

SCimp v2: Group conversation

- Everything encrypted with a single symmetric key

SCimp v2: Group conversation

- Everything encrypted with a single symmetric key
- Group initiator generates a random symmetric key k

SCimp v2: Group conversation

- Everything encrypted with a single symmetric key

- Group initiator generates a random symmetric key k
- Generate random session key ks
- ciphertext $= \text{AES}_{ks}(k)$
- $eks = \text{ECC\_Encrypt}_{pk_B}(ks)$
- Send: eks, ciphertext
- Decrypt

SCimp v2: Group conversation

- Everything encrypted with a single symmetric key

- Group initiator generates a random symmetric key k
- Generate random session key ks
- ciphertext $= \text{AES}_{ks}(k)$
- $eks = \text{ECC\_Encrypt}_{pk_B}(ks)$
- Send: eks, ciphertext
- Decrypt
- Derive group key from k

SCimp v2: Group conversation

- Everything encrypted with a single symmetric key

- Group initiator generates a random symmetric key k
- Generate random session key ks
- ciphertext $= \text{AES}_{ks}(k)$
- $eks = \text{ECC\_Encrypt}_{pk_B}(ks)$
- Send: eks, ciphertext
- Decrypt
- Derive group key from k

- No authentication possible
  - Relies on trust in the server
  - Trivial MITM

SCimp v2: File transfer

- Files are encrypted and uploaded to the cloud
- Keys are exchanged using regular SCimp messages
- Convergent encryption
  - key = hash(file)
  - Missing a salt/secret
  - Vulnerable to file confirmation attack

SCimp v2: File transfer

- Files are encrypted and uploaded to the cloud
- Keys are exchanged using regular SCimp messages
- Convergent encryption
  - key = hash(file)
  - Missing a salt/secret
  - Vulnerable to file confirmation attack
  - Vulnerable to file swapping attack

Proverif results

- First key negotiation (**if** SAS confirmed over authenticated channel)
  - ✓ Confidentiality of keys
  - ✓ Authenticity of keys and other party identity
- Rekeying
  - ✓ Confidentiality of keys
  - ✓ Authenticity of keys and other party identity
  - Future secrecy
    - ✓ When attacker misses first rekeying after compromise
    - ✓ When users reconfirm the SAS
- Sending user message
  - ✓ Confidentiality of keys
  - ✓ Strong secrecy of messages
  - ✓ Authenticity of messages and keys
  - ✓ Forward secrecy (**if** keys can be erased)
  - ✓ Deniability
- Progressive encryption
  - ✗ Confidentiality/authenticity of first message
  - ✓ Confidentiality/authenticity of all messages and keys (**after** SAS is confirmed over an authenticated channel)

Silent Text: SCimp implementation

- CCM implementation does not validate authentication tag
  - Problem in LibTomCrypt (fixed)
- Timing side-channel vulnerability
  - All secrets compared with `memcmp`
- Race condition in message parsing queue
- Message keys are deleted before received messages are validated
- Returned error codes are not checked
- Memory allocation is not checked
- State machine contains bugs and is often bypassed
- Style issues

Comparison with other protocols

	SCimp v1	SCimp v2	OTR	TextSecure
Data in first message	✗	✓(✗)	✗	✓
Key erasure	✓	✓	✓	✓✓
Preshare public keys	✗	✗	✓	✓
Rekey on each reply	✗	✗	✓	✓
Ratchet every message	✓	✓	✗	✓
ECC	✓	✓	✗(✓)	✓

Conclusions

- SCimp version 1 is secure (proven by Proverif)
  - ...but does not solve problems of a mobile environment
- SCimp version 2 solves problems of a mobile environment
  - ...but is insecure
- SCimp implementation has a lot of problems
  - ...lowering both security and user experience
- OTR is secure and good for synchronous environment
- TextSecure is secure and good for mobile environment

Questions?