Aalto University

Network Security: **Formal Verification**

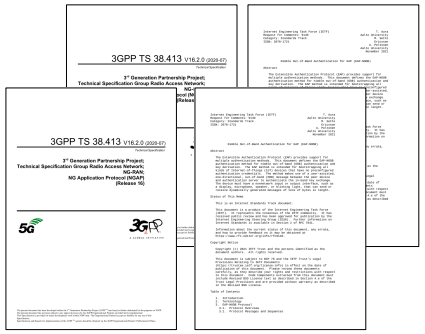
Aleksi Peltonen
CS-E4300 Network security
Aalto University

Formal Verification in Protocol Development

e.g.
IETF
3GPP

Design Protocol

Verify

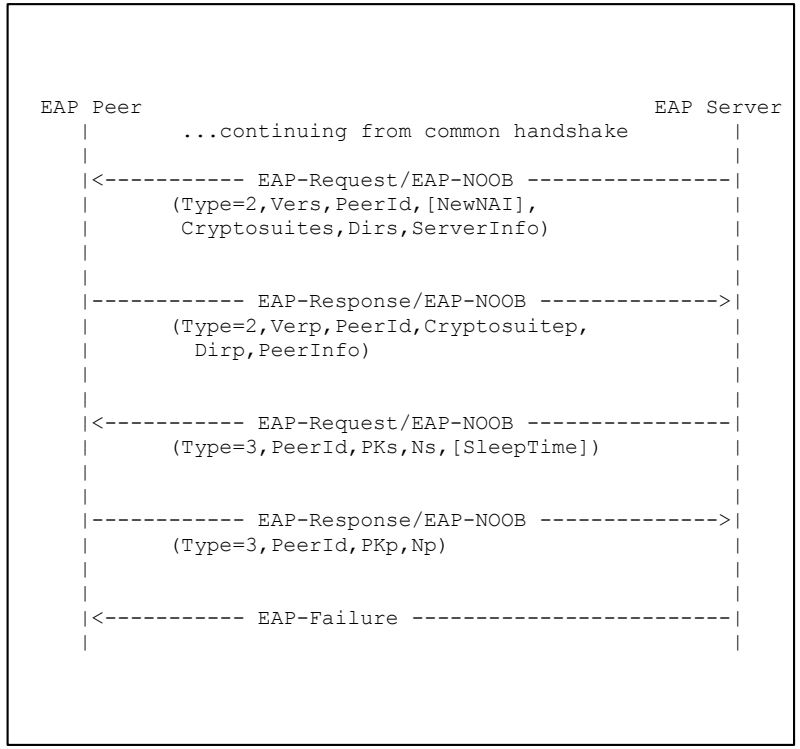


Release Standard

Implement & Deploy

Protocol Modeling

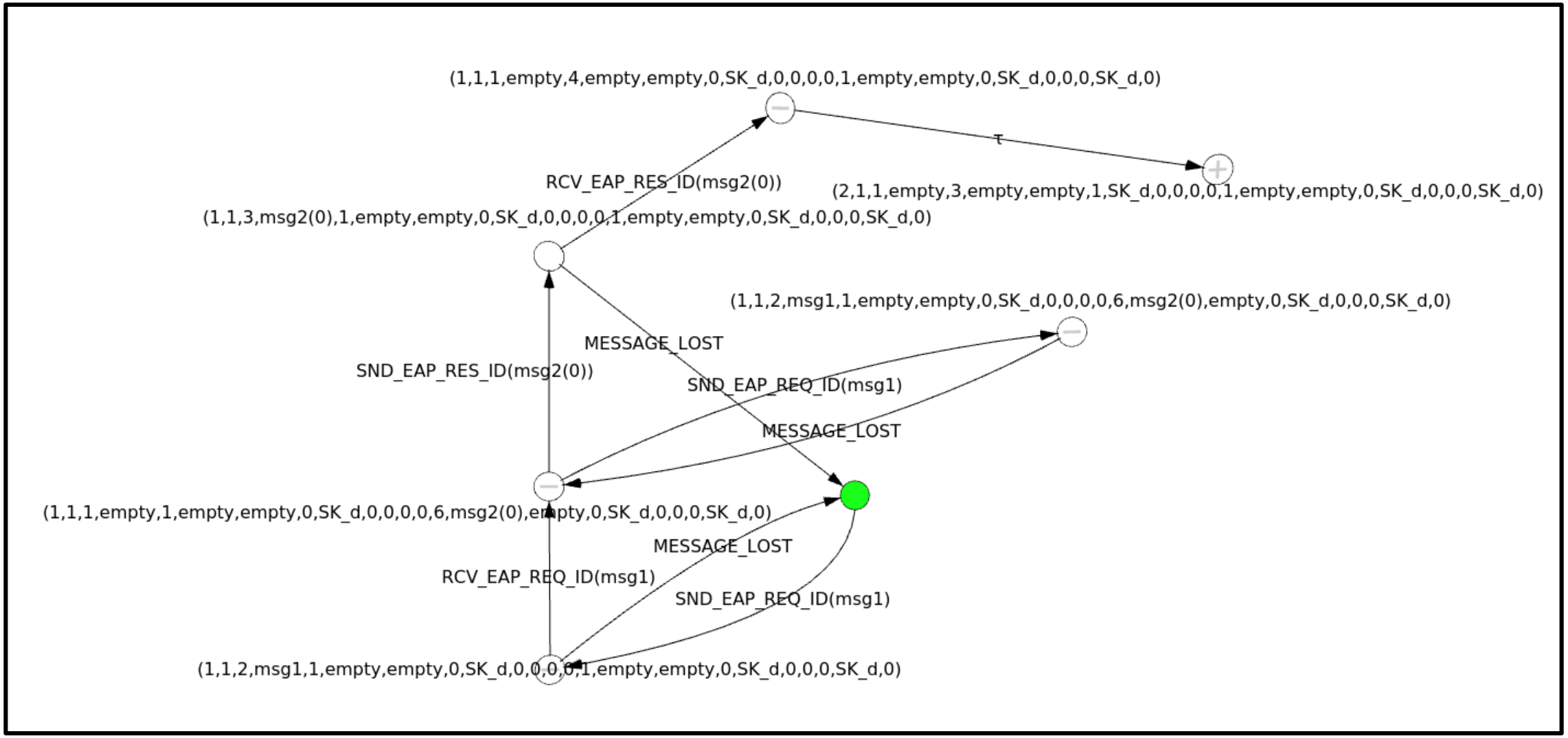
Protocol



Model

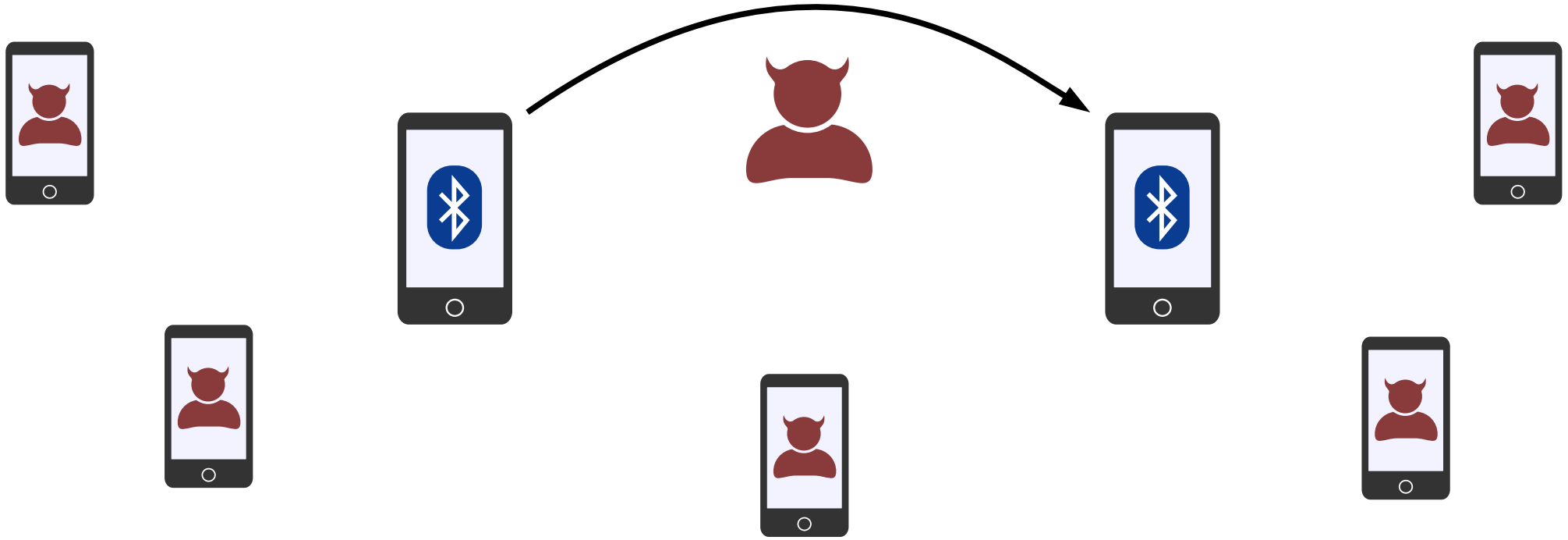
```
(* EAP-Response/Identity *)
in(c, (=NAI));
(* EAP-Request/EAP-NOOB (type 1) *)
out(c, (t1));
(* EAP-Response/EAP-NOOB (type 1) *)
in(c, (=t1,=s0));

(* Generate values *)
new Vers:Ver_l; new PeerId:PeerId_t; new ServerInfo:Info_t;
new Cryptosuites:Cryptosuite_l; new Dirs:Dir_t;

(* EAP-Request/EAP-NOOB (type 2) *)
out(c, (t2,Vers,PeerId,Cryptosuites,Dirs,ServerInfo));
(* EAP-Response/EAP-NOOB (type 2) *)
in(c, (=t2,Verp:Ver_t,=PeerId,Cryptosuitep:Cryptosuite_t,
       Dirp:Dir_t, PeerInfo:Info_t));

(* Generate nonce *)
new Ns:N_t;
(* Server public key *)
let PKs = pk(SKs) in

(* EAP-Request/EAP-NOOB (type 3) *)
out(c, (t3,PeerId,PKs,Ns));
(* EAP-Response/EAP-NOOB (type 3) *)
in(c, (=t3,=PeerId,PKp:K_t,Np:N_t));

(* EAP-Failure *)
out(c, EAP_Failure);
```

Protocol Modeling



Dolev-Yao Attacker

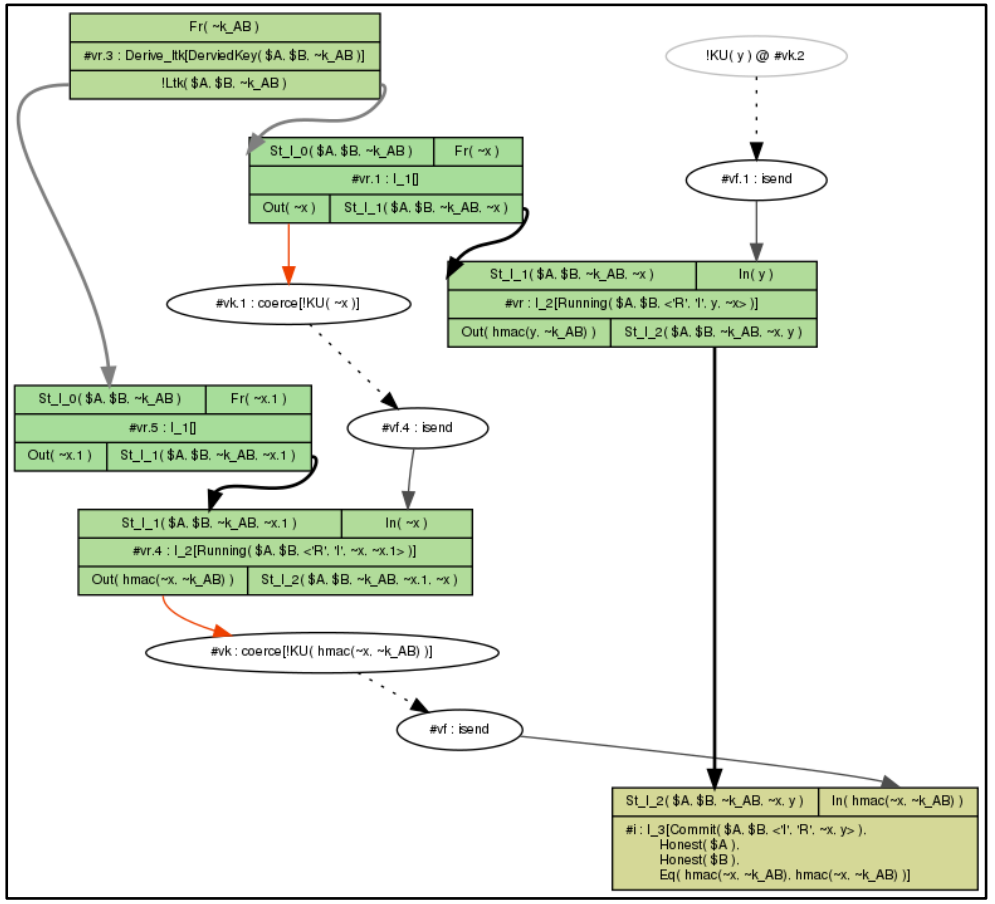
- Attacker **can** read, modify, delete, and inject messages.
- Attacker **can not** decrypt messages without encryption keys.



Use-cases

- Formal verification is useful for..
  - ..finding attacks in large protocols.
  - ..spotting mistakes in your design.
  - ..finding variations of attack traces.

- Formal verification **does not**..
  - ..prove correctness of the protocol.
  - ..find implementation issues.



Case study:

Misbinding in device pairing with Bluetooth

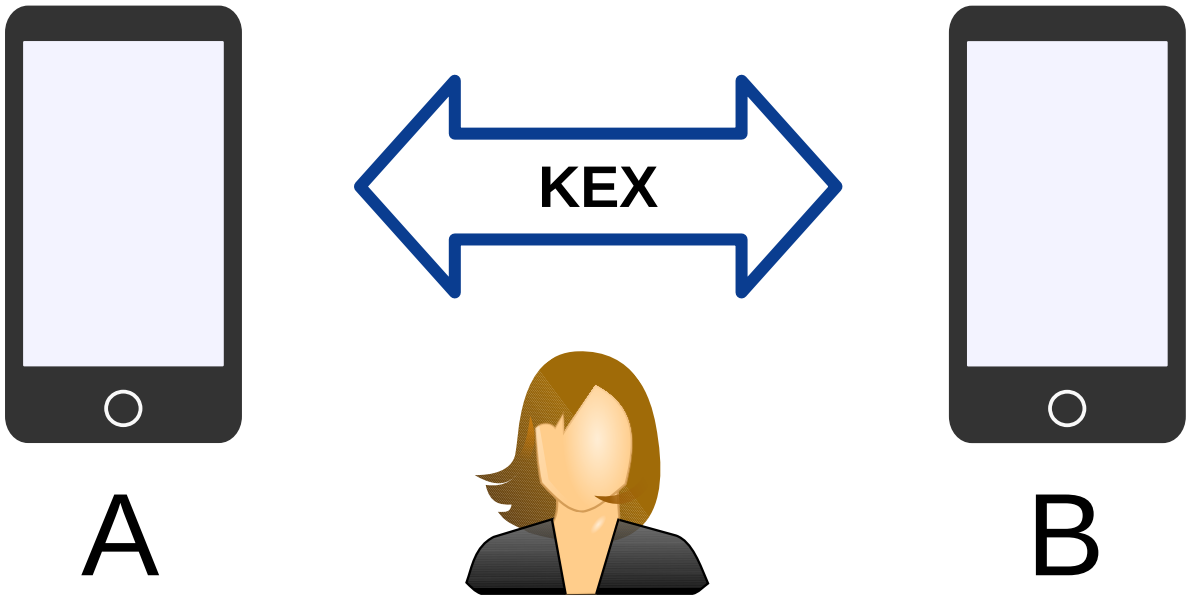
Sethi M, Peltonen A, Aura T. Misbinding attacks on secure device pairing and bootstrapping. In: Proceedings of the 2019 ACM Asia conference on computer and communications security, Asia CCS ’19. ACM, New York; 2019. p. 453–464. https://doi.org/10.1145/3321705.3329813.

Bluetooth numeric comparison

1. Make device B discoverable
2. On device A, search and select B
3. Key exchange in background
4. Compare 6-digit codes and press OK → Paired!

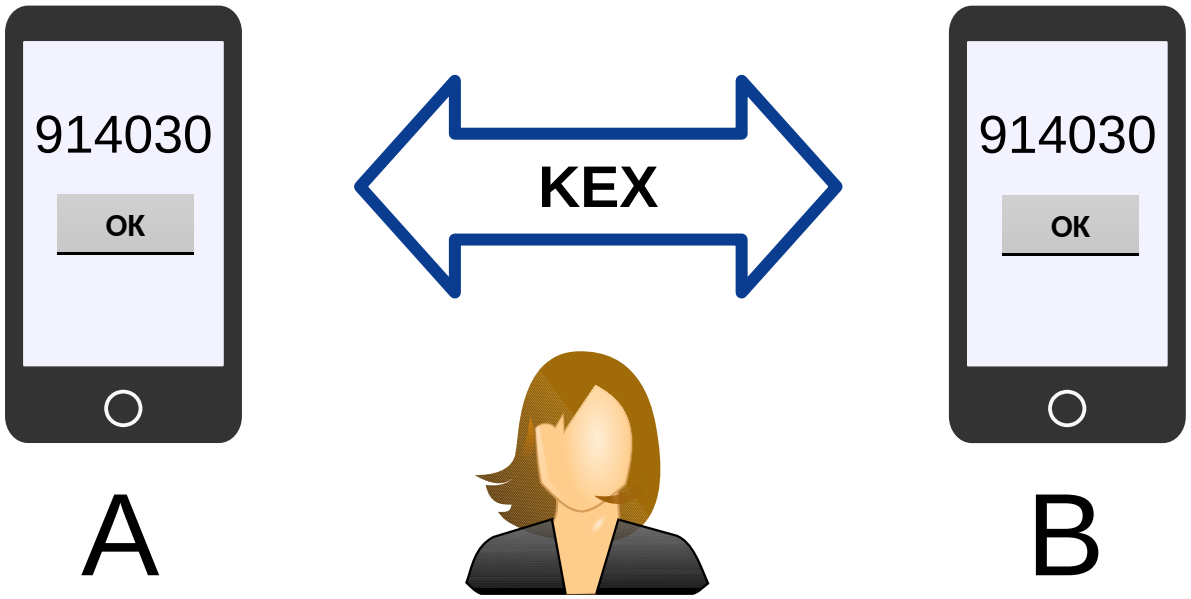
Bluetooth numeric comparison

1. Make device B discoverable
2. On device A, search and select B
3. Key exchange in background
4. Compare 6-digit codes and press OK → Paired!



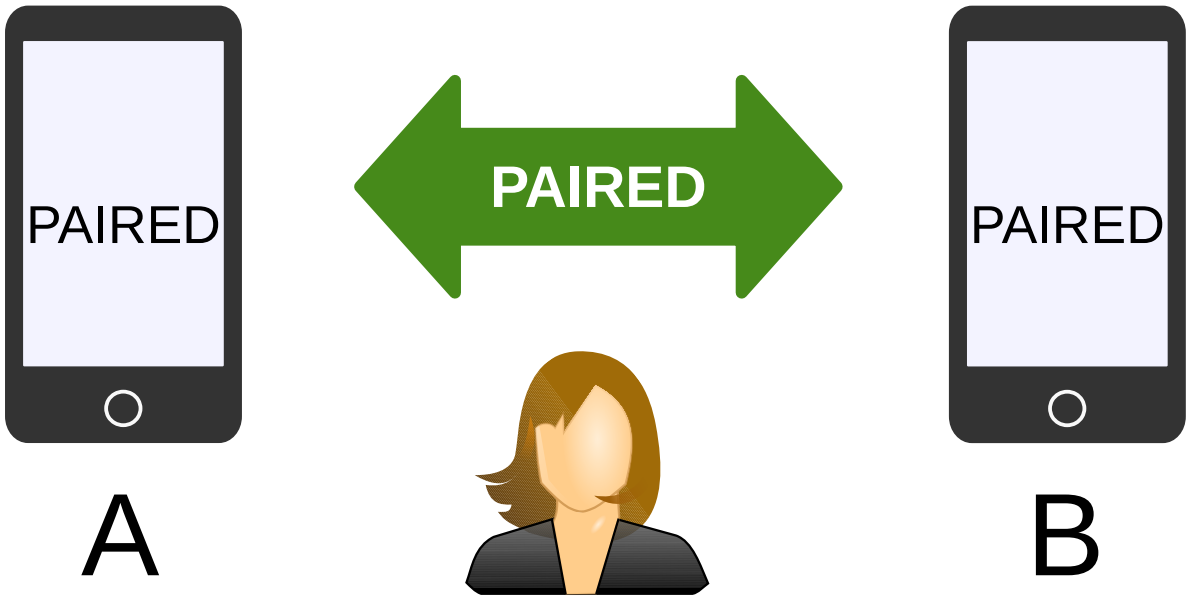
Bluetooth numeric comparison

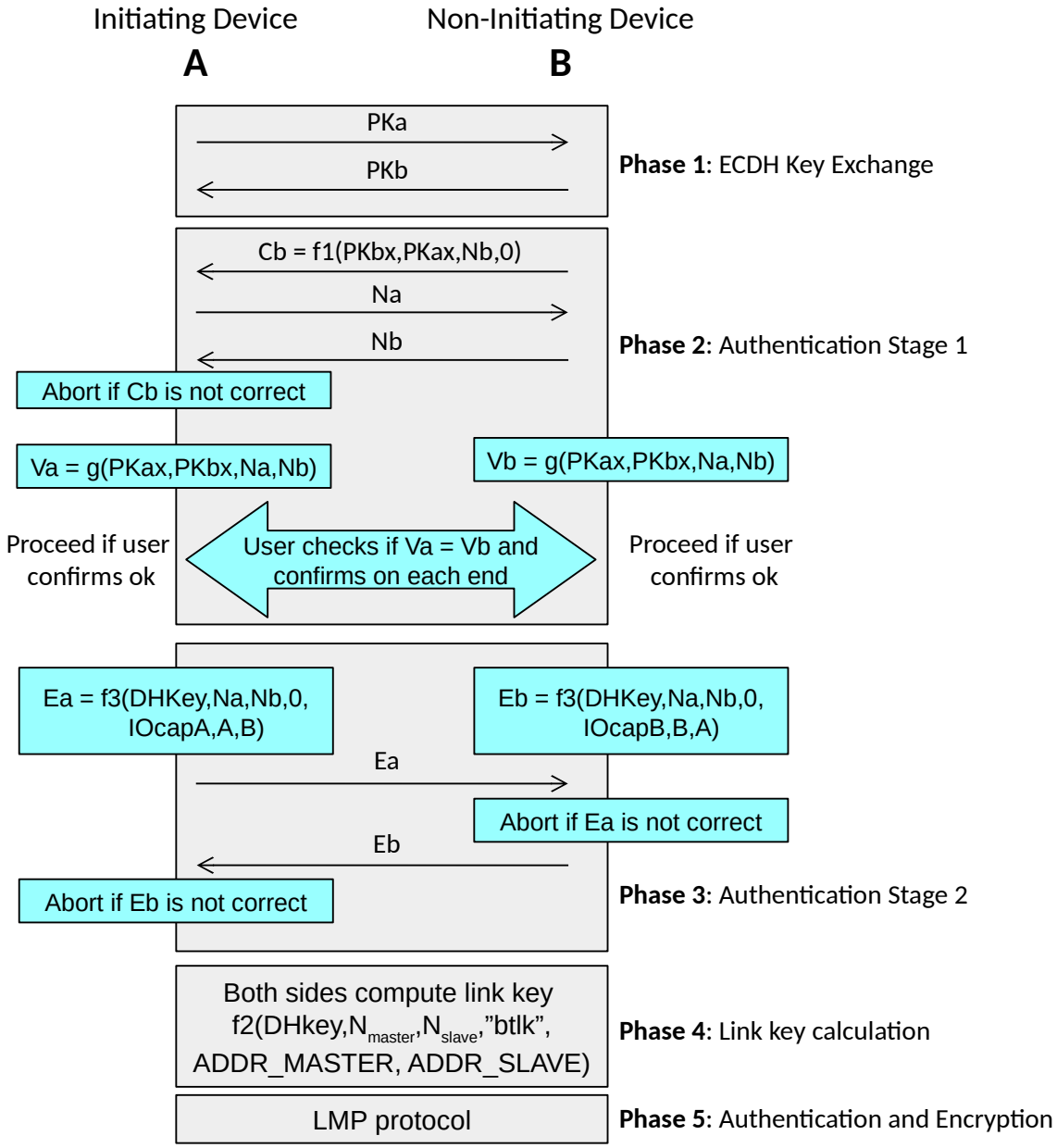
1. Make device B discoverable
2. On device A, search and select B
3. Key exchange in background
4. Compare 6-digit codes and press OK → Paired!



Bluetooth numeric comparison

1. Make device B discoverable
2. On device A, search and select B
3. Key exchange in background
4. Compare 6-digit codes and press OK → Paired!




Initiating Device
A
Non-Initiating Device
B
PKa
PKb
Phase 1: ECDH Key Exchange
Cb = f1(PKbx,PKax,Nb,0)
Na
Nb
Phase 2: Authentication Stage 1
Abort if Cb is not correct
Va = g(PKax,PKbx,Na,Nb)
Vb = g(PKax,PKbx,Na,Nb)
Proceed if user confirms ok
User checks if Va = Vb and confirms on each end
Proceed if user confirms ok
Ea = f3(DHKey,Na,Nb,0, IOcapA,A,B)
Eb = f3(DHKey,Na,Nb,0, IOcapB,B,A)
Ea
Abort if Ea is not correct
Eb
Abort if Eb is not correct
Phase 3: Authentication Stage 2
Both sides compute link key
f2(DHkey,N_master,N_slave,"btlk", ADDR_MASTER, ADDR_SLAVE)
Phase 4: Link key calculation
LMP protocol
Phase 5: Authentication and Encryption

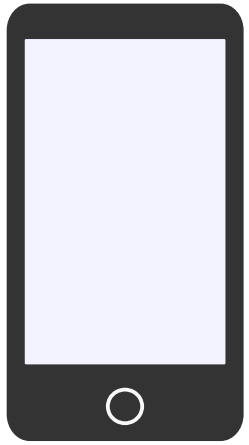
Demo:

Discovering the misbiding attack with ProVerif

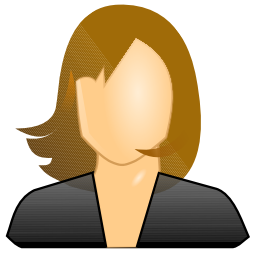
ProVerif

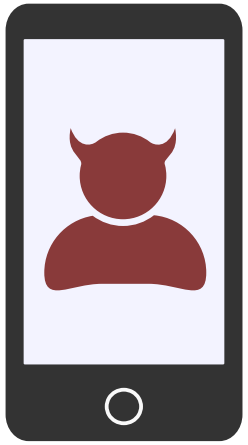
- ProVerif is a tool for modeling and automatic verification of cryptographic protocols and their security goals.
- It can be used for proving secrecy and authentication properties.
- ProVerif analyzes the protocol over an unbounded number of sessions and messages. It tries to construct an attack trace when the target property fails.
  - Results are either **true**, **false**, or **undecided.**
- Models are written in the typed pi calculus and can be divided into three parts:
  1. **Declarations** formalize the behavior of cryptographic primitives.
  2. **Process macros** allow sub-processes to be defined, in order to ease development.
  3. A **main process**, which using macros encode the protocol itself.

Misbinding in Bluetooth



A

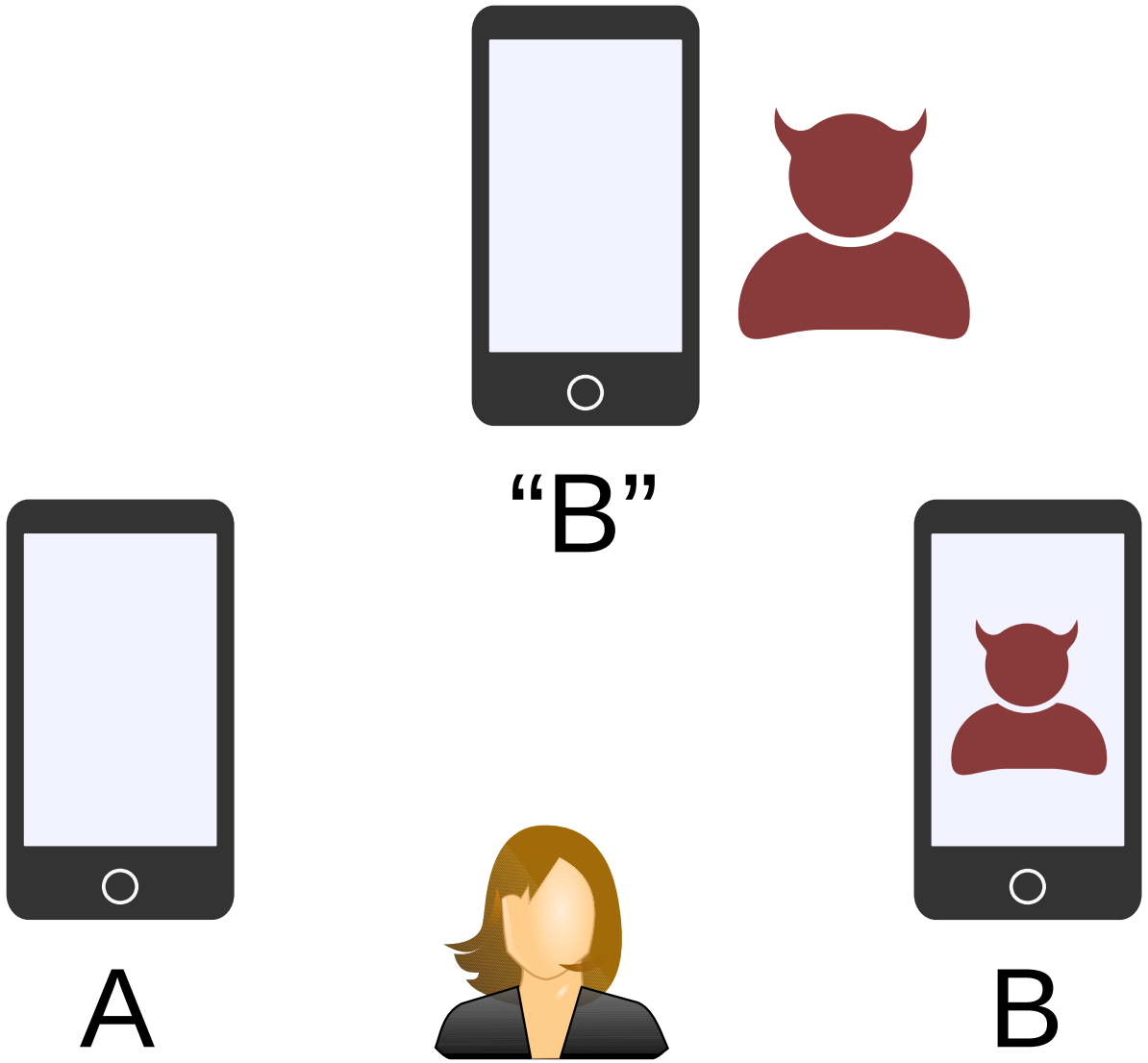




B

Adversary has limited control of B (Malicious app)

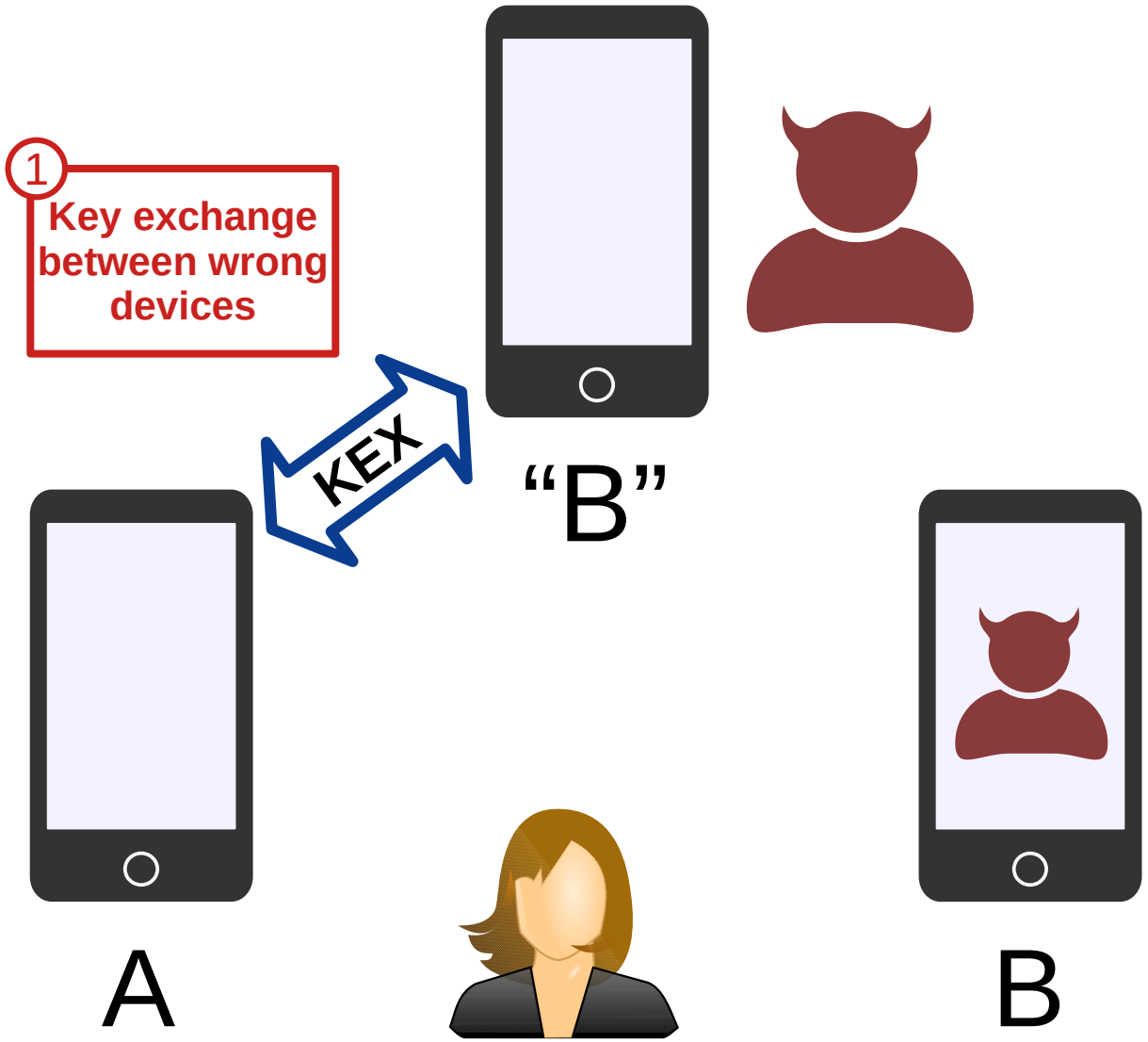
Misbinding in Bluetooth



Attacker has another device named “B”

Adversary has limited control of B (Malicious app)

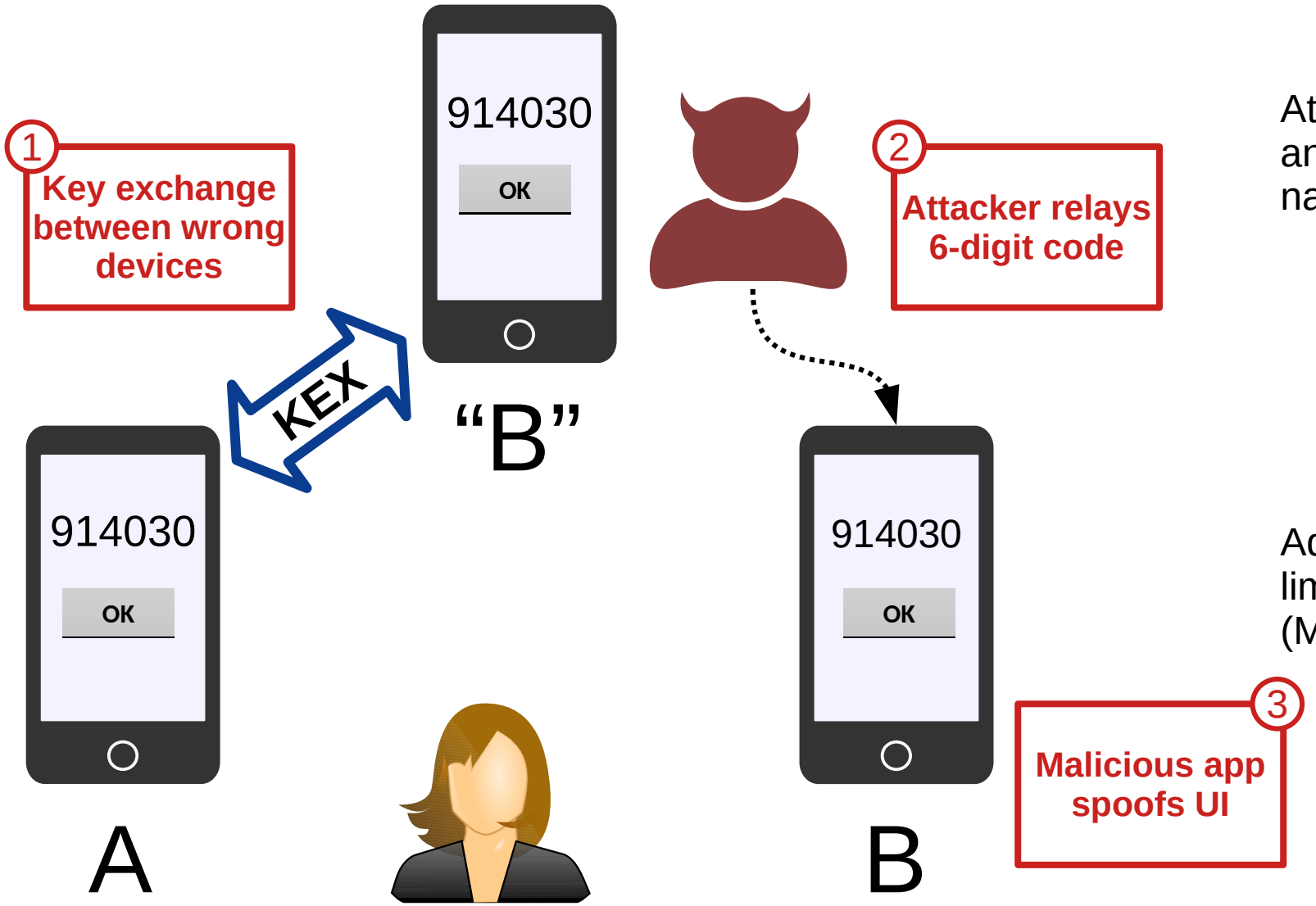
Misbinding in Bluetooth



Attacker has another device named “B”

Adversary has limited control of B (Malicious app)

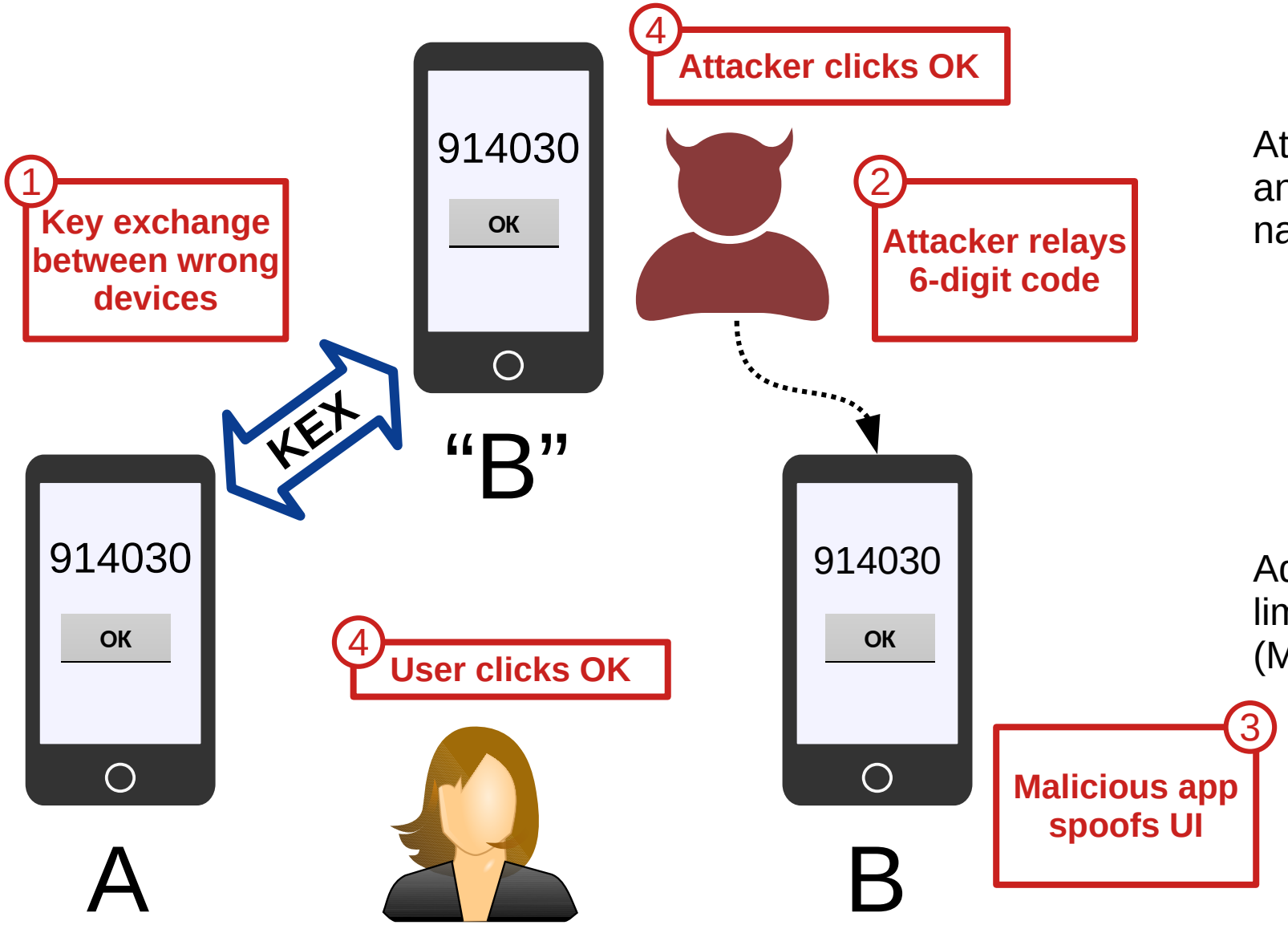
Misbinding in Bluetooth



Attacker has another device named "B"

Adversary has limited control of B (Malicious app)

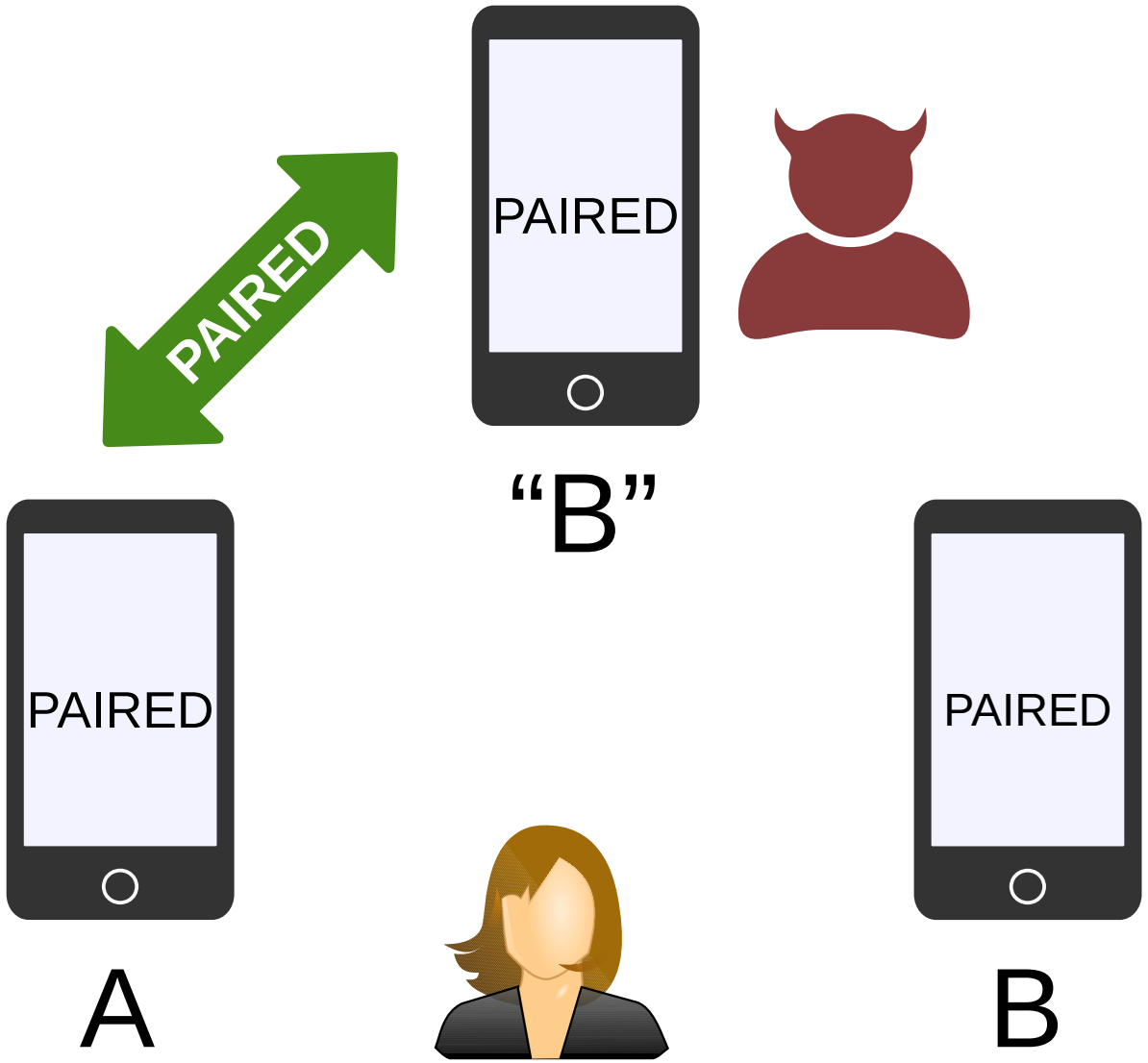
Misbinding in Bluetooth



Attacker has another device named "B"

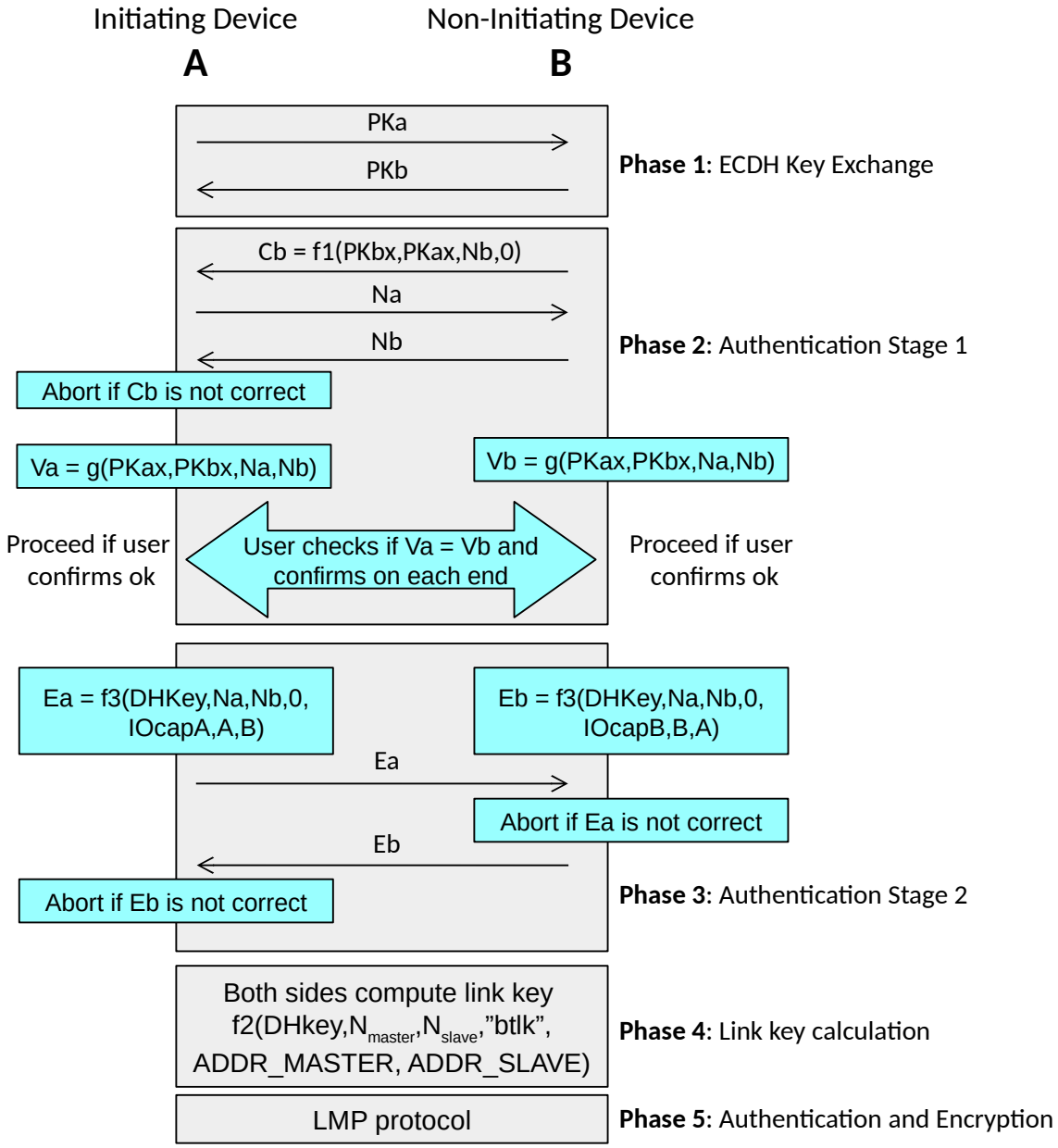
Adversary has limited control of B (Malicious app)

Misbinding in Bluetooth



Attacker has another device named “B”

Adversary has limited control of B (Malicious app)



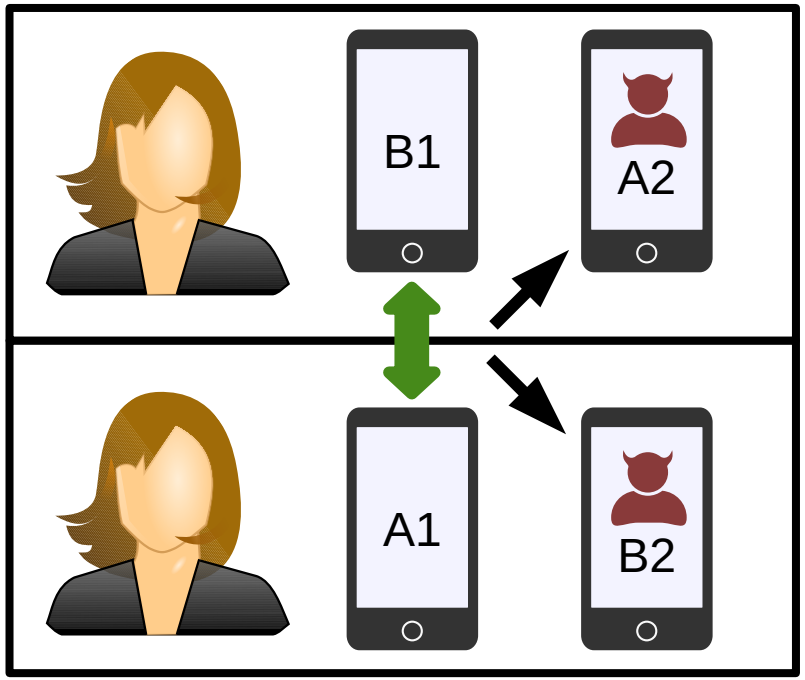
- Why does Bluetooth not detect misbinding?
- Could it?
- Devices have no verifiable identifiers!
- Authentication based only on physical access

Formal Verification of Bluetooth

- Previous security analysis of Bluetooth had not detected misbinding
- We modeled Bluetooth numeric comparison and other pairing protocols with ProVerif
  - Physical channel defines device identity
  - Check correspondence between user intentionand completed pairing

  → Can detect misbinding

- Analysis yielded a new double-misbinding case



Five Variations of the Misbinding Attack