

The Candidate Key Protocol for Generating Secret Shared Keys from Similar Sensor Data Streams

ESAS 2007
2. July 2007, 09:00

Rene Mayrhofer
Lancaster University, UK

Motivation
Protocol
Implementation
Discussion

Device pairing/association
DH-based approaches
Why yet another protocol?

The scenario: spontaneous authentication



What do we already have?

Current approaches:

- Two (or multiple) phases
- **Key agreement**: typically select peer device + Diffie-Hellman
- **Peer authentication**: various options
 - commitment schemes
 - interlock-based protocols
- Verification based on some **out-of-band channel**
 - verification of key hashes: display+user+yes/no
 - transmission over secret and/or authentic channel: display+user+keypad, infrared, ultrasound, laser, display+camera, audio, NFC, ...
 - shared secret: common data, possibly “fuzzy”

Problem of scalability

Most current protocols scale linearly with number of potential neighbors

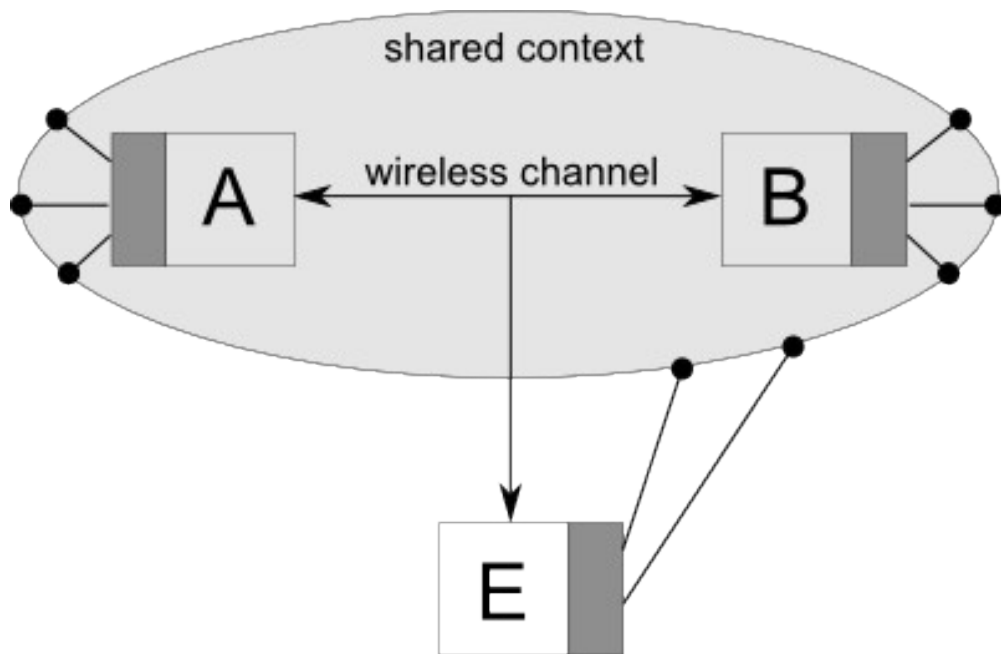
- already good in terms of network, CPU, power consumption, etc.
- but: user attention does not scale as well

User interaction is problematic

- explicit selection of network entity *or*
- only selection of physical entity with implicit binding to network entity
⇒ (slight) delay while multiple devices are tried

Better: **constant-scalable interaction** with broad-/multicast protocols

Context-based authentication



- main threat scenario: MITM on wireless communication channel
- intended communication partners A and B share some context
- attacker E has inferior access to this context
- respective aspect of context represented by sensor data streams
⇒ shared (**weakly**) secret information

From sensor data to keys: first steps

1. Sensor data acquisition

- potential problem: side-channel attacks

2. Temporal alignment

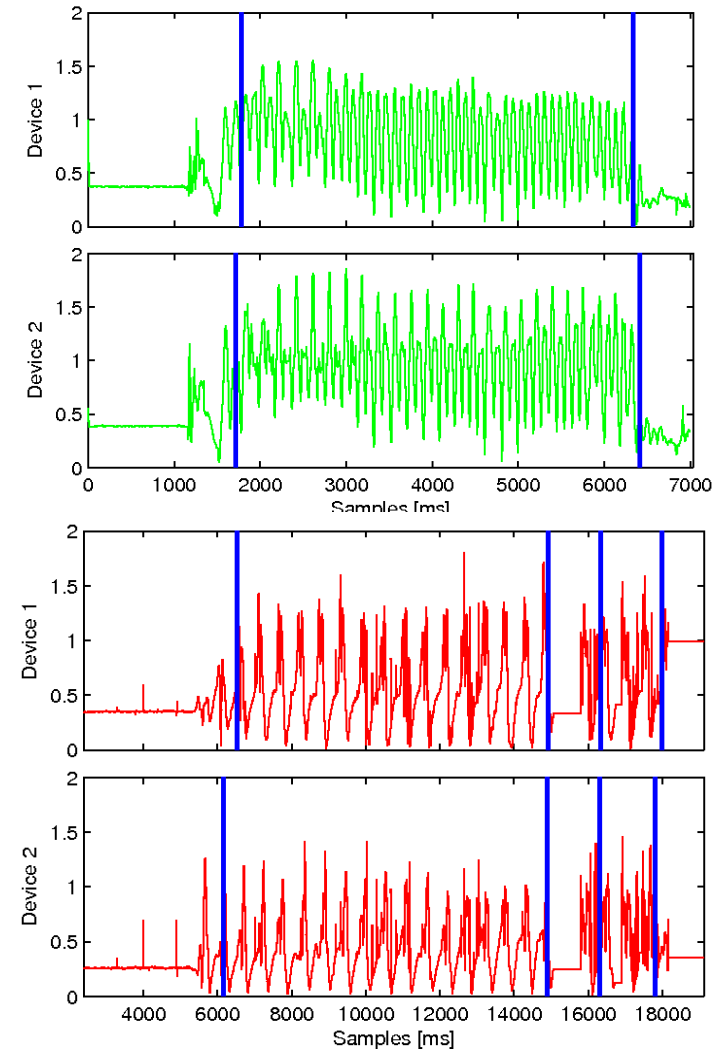
- triggering
- synchronization

3. Spatial alignment

- when using multi-dimensional input data, these dimensions may be in different reference systems on different devices

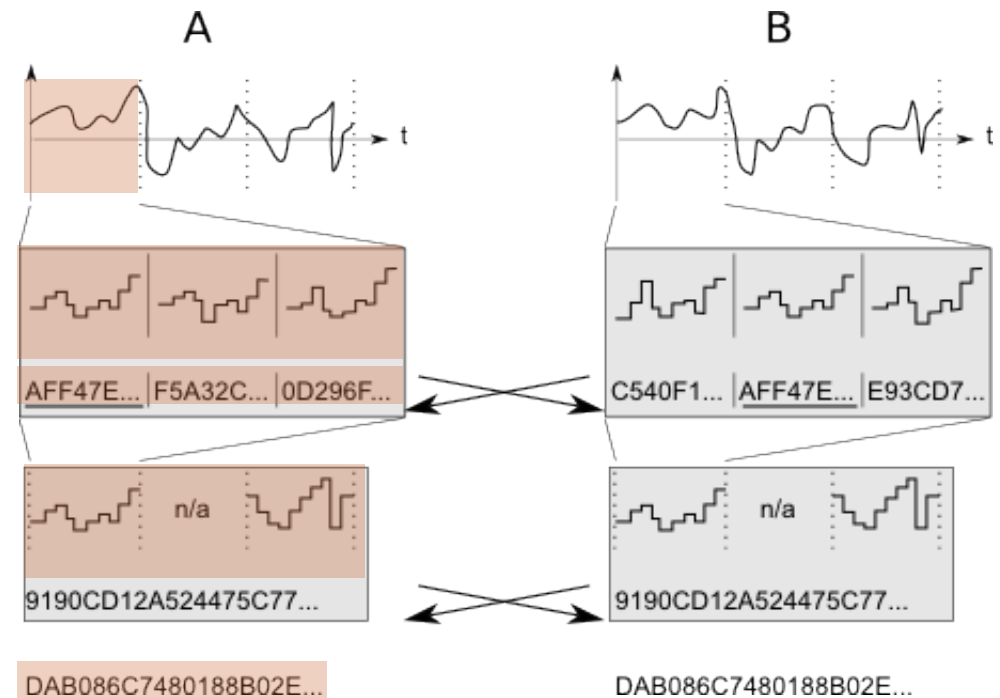
4. Feature extraction

- raw data usually unusable, domain-specific

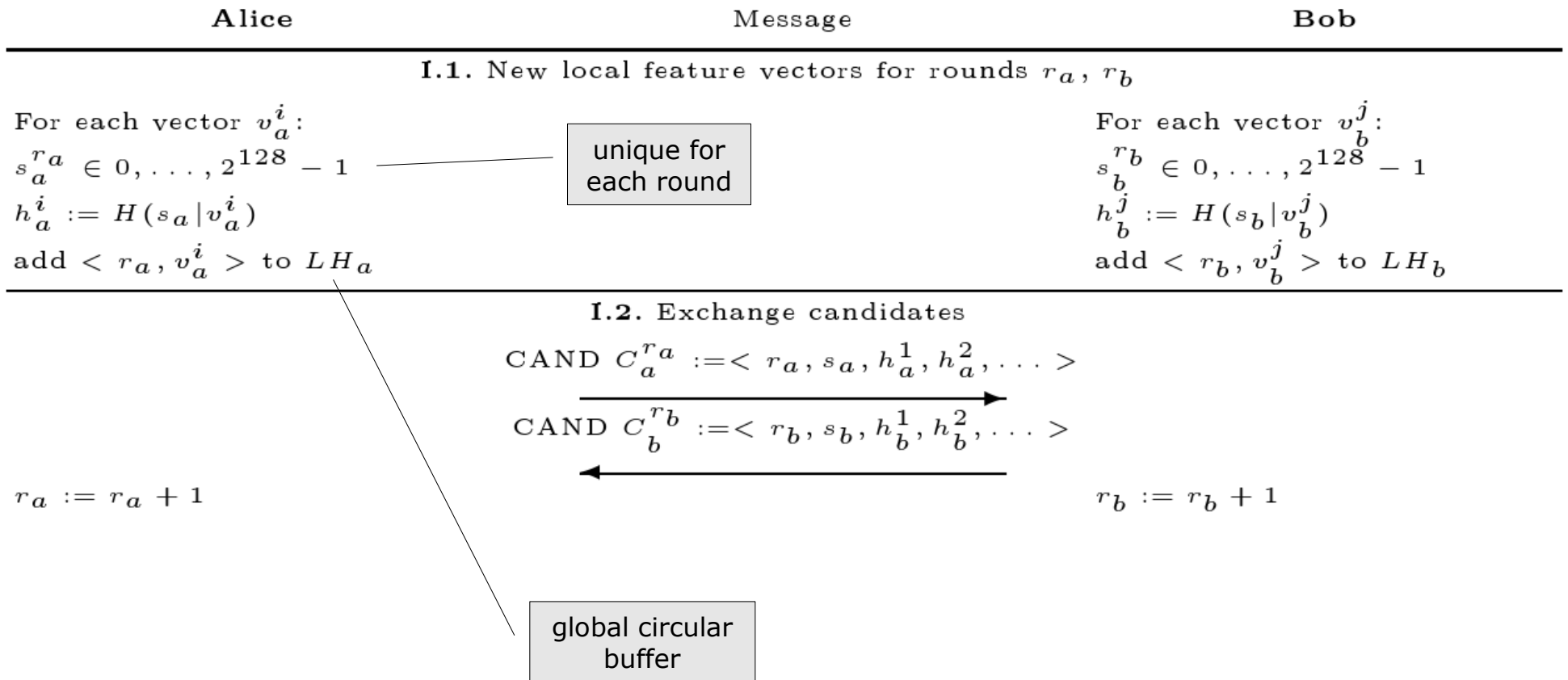


The Candidate Key Protocol

1. features extracted from sensor data streams
⇒ multiple feature vectors, e.g. with different quantization offsets, different parameter sets, etc.
2. multiple **candidate key parts** for each time window, hashes are **broadcast**
3. multiple **matching key parts** concatenated to **candidate keys**, hashes are sent to respective host
4. when candidate key can be generated at receiving host, acknowledge and use as **shared secret key**



A bit more detail...



A bit more detail...

I.3. New remote candidates

For each hash \tilde{h}_b^j :
 if $\exists \langle \hat{r}_a, \hat{v}_a^i \rangle \in LH_a$
 s.t. $\tilde{h}_b^j = H(s_b | \hat{v}_a^i)$

then add j to $ML_a^{r_b}$
 and add $\langle \hat{r}_a, \hat{v}_a^i \rangle$ to $MC_{a,b}$

remember *remote*
 round/candidate numbers

specific to
 remote host

For each hash \tilde{h}_a^i :
 $\exists \langle \hat{r}_b, \hat{v}_b^j \rangle \in LH_b$
 s.t. $\tilde{h}_a^i = H(s_a | \hat{v}_b^j)$

then add i to $ML_b^{r_a}$
 and add $\langle \hat{r}_b, \hat{v}_b^j \rangle$ to $MC_{b,a}$

I.4. (optional) Exchange matches

MATCH $M_a^{r_a} := \langle \tilde{r}_b, ML_a^{r_b} \rangle$

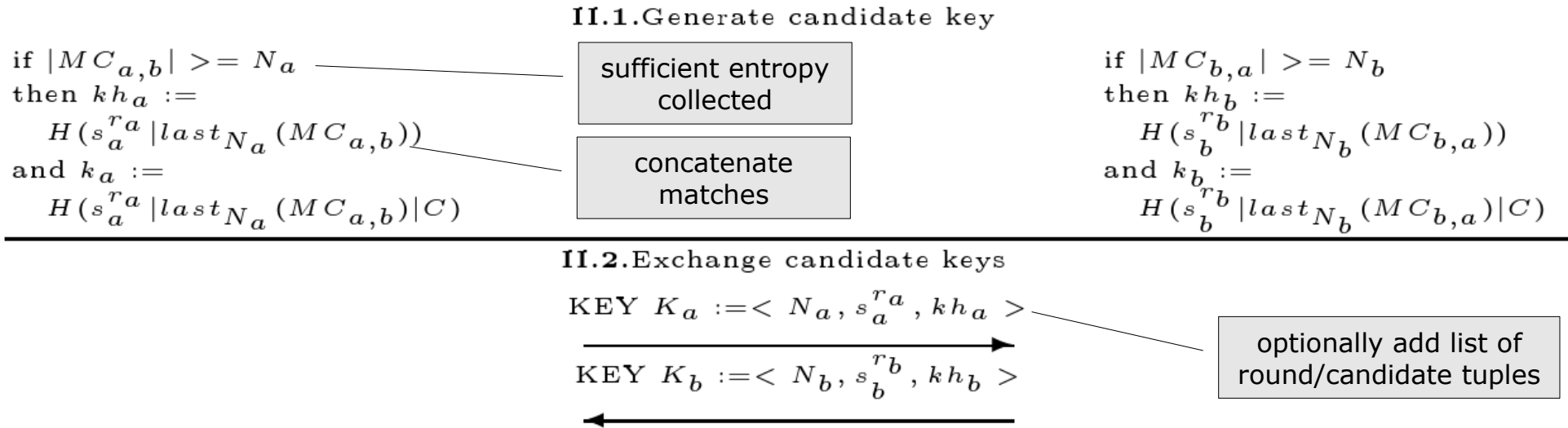
MATCH $M_b^{r_b} := \langle \tilde{r}_a, ML_b^{r_a} \rangle$

only for asymmetrical
 settings

add numbers $ML_b^{r_a}$ of
 round \tilde{r}_a from LH_a to $MC_{a,b}$

add numbers $ML_a^{r_b}$ of
 round \tilde{r}_b from LH_b to $MC_{b,a}$

A bit more detail...



A bit more detail...

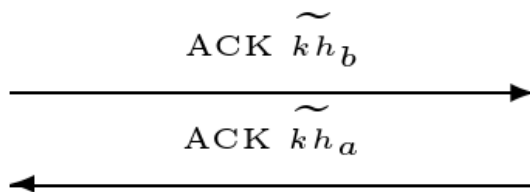
II.3. Search for candidate key

if $\exists \tilde{c}_a \subseteq MC_{a,b}$
 s.t. $H(\tilde{s}_b^{r_b} | \tilde{c}_a) = \tilde{k}h_b$
 then $k'_b := H(\tilde{s}_b^{r_b} | \tilde{c}_a | C)$

search because of
 multiple matches: $O(A^H)$

if $\exists \tilde{c}_b \subseteq MC_{b,a}$
 s.t. $H(\tilde{s}_a^{r_a} | \tilde{c}_b) = \tilde{k}h_a$
 then $k'_a := H(\tilde{s}_a^{r_a} | \tilde{c}_b | C)$

II.4. Acknowledge key



II.5. Set key

if k_a is not set $k := k'_b$
 if k'_b is not set $k := k_a$
 if $\tilde{k}h_a = \tilde{k}h_b$
 then $k := k_a$
 else $k := k_a \oplus k'_b$

messages can
 overlap "in-flight"

if k_b is not set $k := k'_a$
 if k'_a is not set $k := k_b$
 if $\tilde{k}h_b = \tilde{k}h_a$
 then $k := k_b$
 $k := k_b \oplus k'_a$

And how to implement it?

From abstract channels to the real world

- Channels like ZigBee, WLAN are **lossy**
- UDP has comparable guarantees to such low-level broadcast channels

Effects of losing messages:

- **CAND** (candidate key parts) \Rightarrow can not be used as matches
- **MATCH** (match acknowledgments) \Rightarrow same as losing CAND
- **KEY** (candidate keys) \Rightarrow when no acknowledgement received, next KEY
- **ACK** (key acknowledgments) \Rightarrow same as losing KEY

Asynchronism

- general case: message received before local round processed \Rightarrow message history replay

What we missed on the first N tries

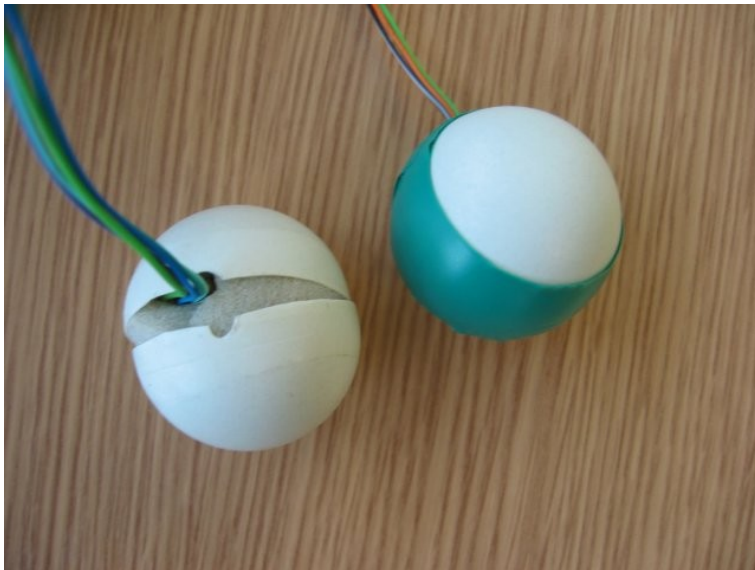
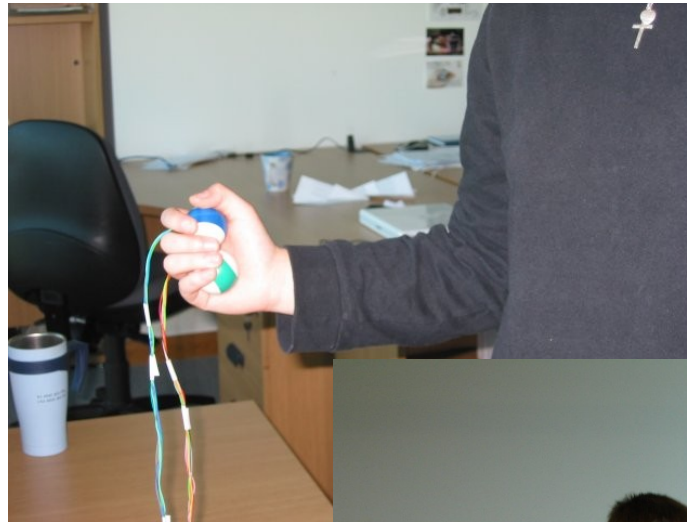
- Asynchronism (the “back to the future” problem)
- Checking for matches can't stop at the first match (the “early optimization” and the “the remote side may match the other way around” problems)
- Transmitting round/candidate tuples (the “but sometimes need to optimize” problem)
- Local synchronisation (the “oh, the remote talks at inconvenient times” and “now we're safe, but deadlocked” problems)
- Complete message buffers (the “how can we have a KEY message before its last CAND message?” problem)
- Exponential search complexity (the “don't use integers for computing factorials” problem)
- UDP sockets peculiarities (the “under XYZ, UDP multicast doesn't loop back” and “oh, a host can have multiple interfaces?” problems)

How we caught them

- Sanity checks
- Sanity checks (really!)
- Sanity checks (there's no such thing as "that variable **will** be set at this place")
- Unit tests
- Write test cases **before** using a new class/method somewhere else (not quite as rigid as test-first, but highly valuable)
- Write test cases for cases you **don't** want to happen ("that will **never** be the case in practice" - right...)

Does it work?

- 2x 2D accelerometer per "device": ADXL202JE
- Sampled directly with the parallel port

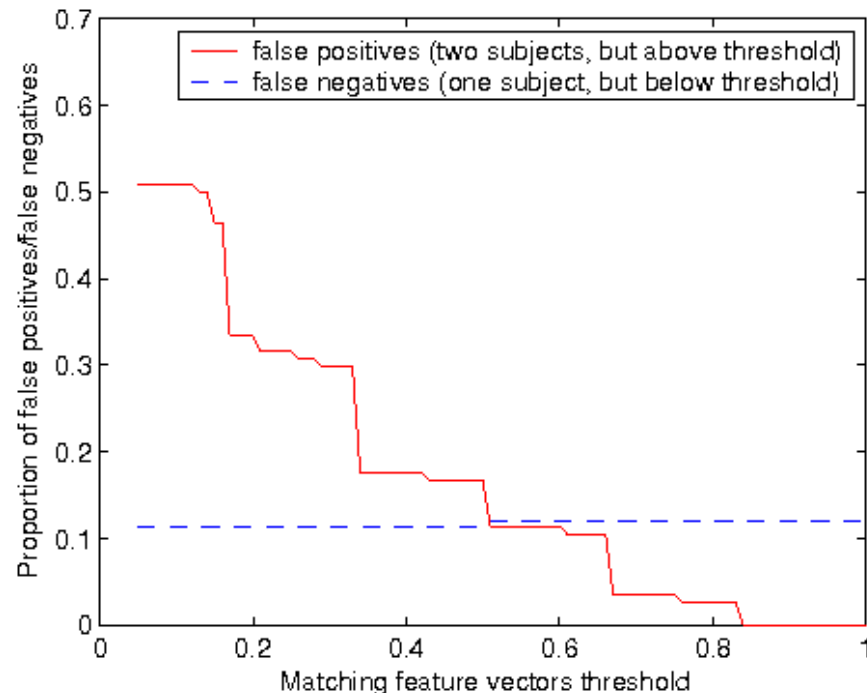


[MaGe 2007b] R. Mayrhofer, H. Gellersen: "Shake well before use: Authentication based on accelerometer data", Pervasive 2007

Does it work?

3 experiments:

- How do people shake?
- "Hacking" competition
- Live mode - does it work?



Results:

- Parameters for **no false positives**
- False negatives 11.96%
- 27/30 subjects reproducibly successful within 10 tries or less
- Entropy of each feature vector estimated at ~ 7 bit

The bad

- **Offline lookup table attacks** possible when feature vectors have insufficient entropy
- CKP can not increase entropy of key material (related work suggests that entropy is at least retained)

The good

- Single, continuous phase
- Devices “**tune into**” each other's key streams
- **Implicit** authentication
- **Opportunistic** authentication
- **Multi-device** authentication
- Applicable to arbitrary feature vectors
- Open source (LGPL) reference implementation - “it's there”

The ugly

- Do MATCH messages reveal additional information?
- Can we increase the entropy by adding (more) random input (privacy amplification)?
- How best to counter attacks on asynchronism?

Summary: What have we done?

- Candidate Key Protocol **interactively** creates shared secret keys from **common sensor data** streams
- Multiple **candidates** for each time window reduce problems with noise
- **Broadcasting hashes** to find matches without revealing them
- Remote devices “**tune into**” candidate key stream
 - resource friendly
 - opportunistic
 - multi-device
- Implementation under LGPL at <http://www.openuat.org>



“plus valet in manibus avis unica quam
dupla silvis”

Ca. 1200AD

A bird in the hand is worth two in the bush.



IWSSI 2007

First International Workshop on Security for Spontaneous Interaction at UbiComp 2007

Thank you for your attention!

Slides: <http://www.mayrhofer.eu.org/presentations>

Source code: <http://www.openuat.org>

Later questions: rene@mayrhofer.eu.org

OpenPGP key: 0xC3C24BDE

7FE4 0DB5 61EC C645 B2F1 C847 ABB4 8F0D C3C2 4BDE