Virtual Smart Cards:
How to Sign with a Password and a Server

Jan Camenisch[1], Anja Lehmann[1], Gregory Neven[1], Kai Samelin[1,2]

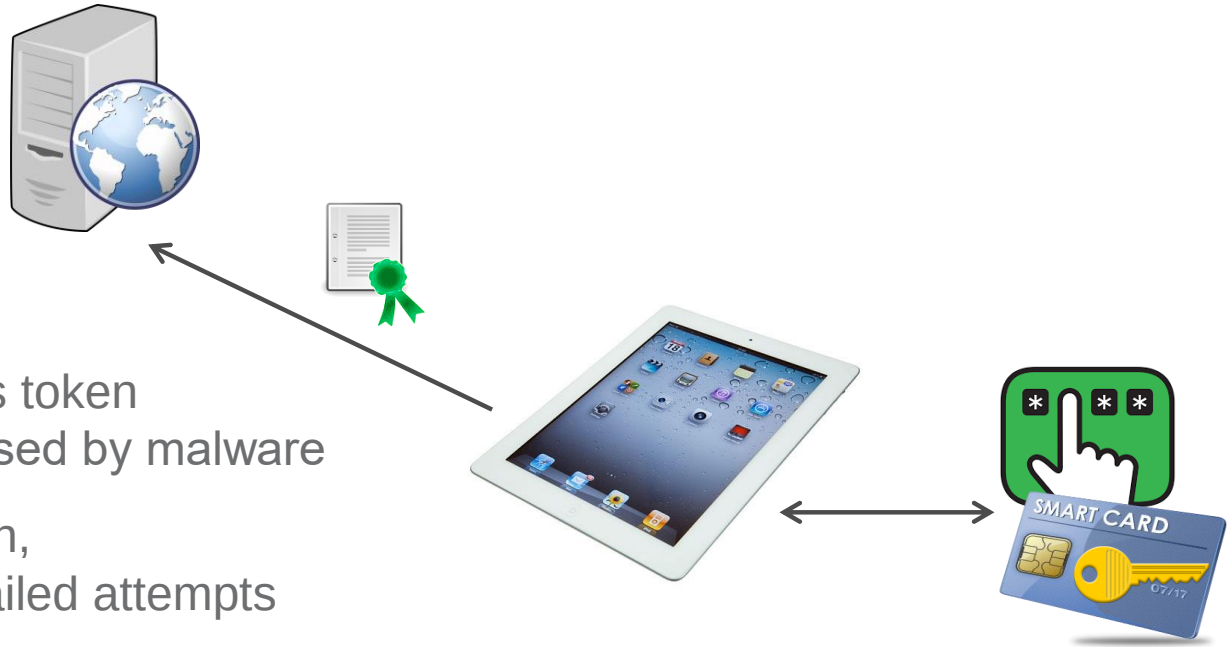
[1] IBM Research – Zurich
[2] Technische Universität Darmstadt

IBM

Protecting users’ crypto keys

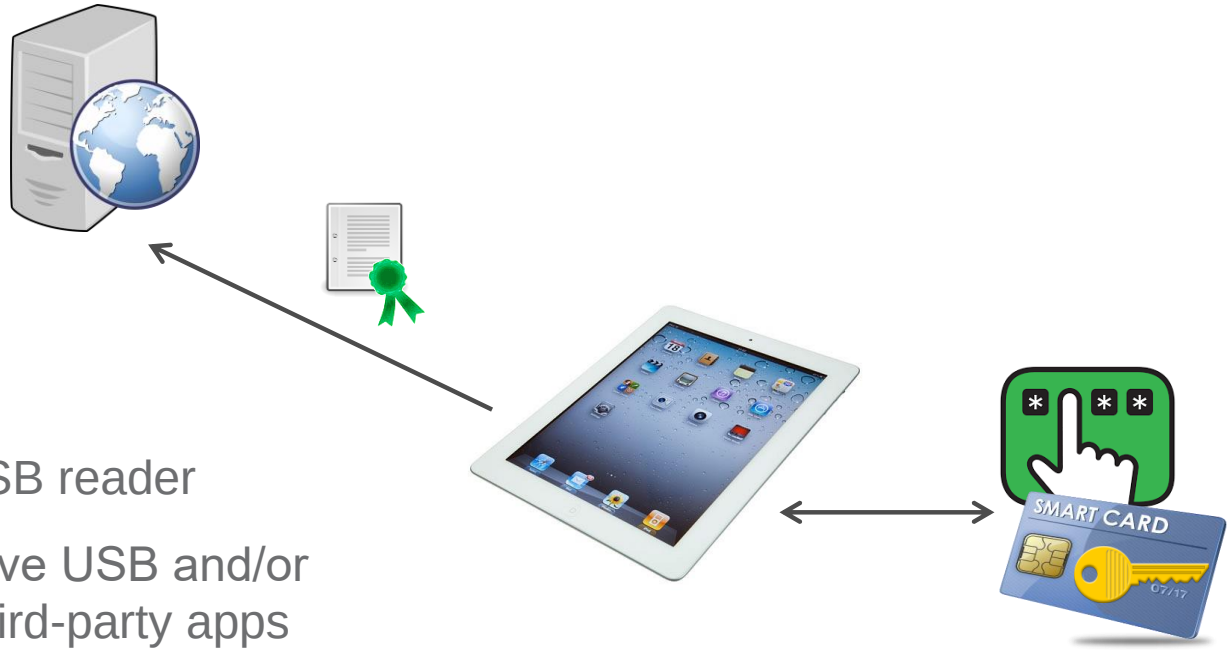
- Password-encrypted storage
  offline dictionary attacks
- Secret sharing
  locally reconstructed keys are vulnerable to malware
- Trusted hardware tokens
  e.g., smartcard, TPM, secure element

The case for hardware tokens



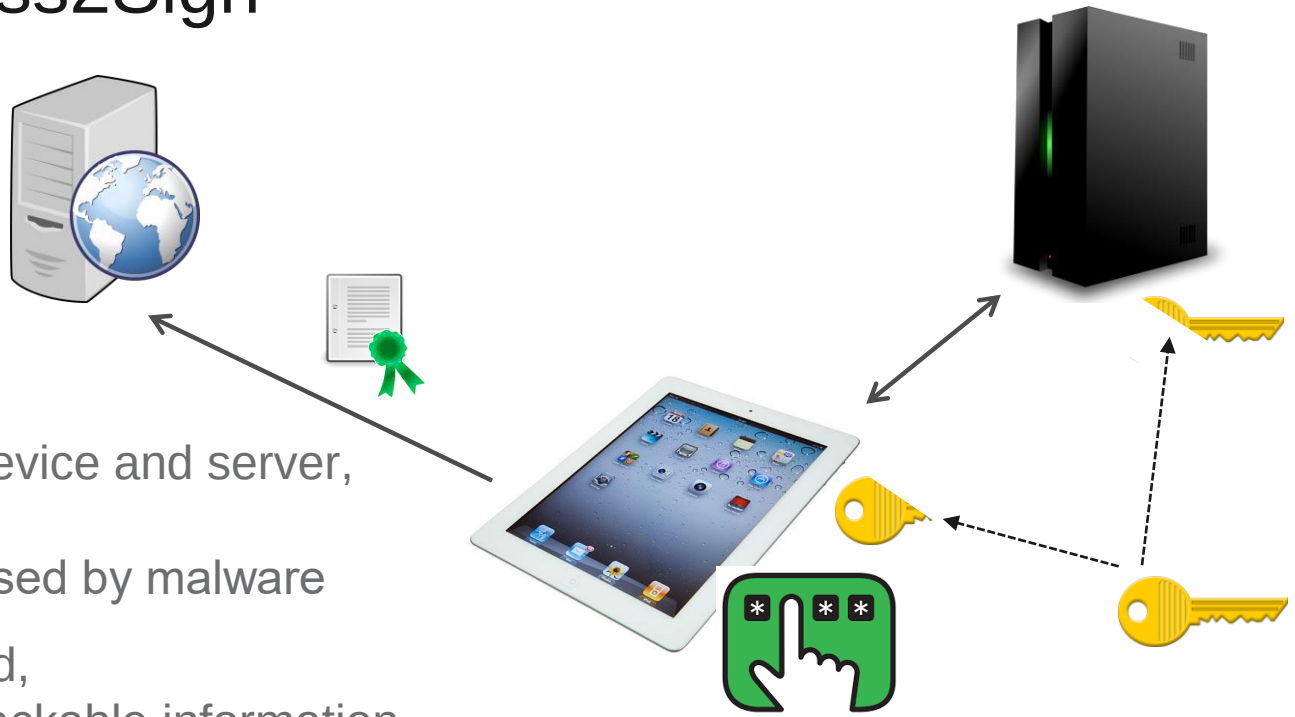
- Secret key never leaves token
  → cannot be compromised by malware
- PIN/password protection,
  blocks after too many failed attempts
- ~~Token only signs messages approved by user~~ (only with dedicated input/display)
  Token cannot sign when unplugged
- Stolen or lost → revoke certificate

The problem with hardware tokens



- Inconvenient to carry
- May require external USB reader
- Mobile devices don’t have USB and/or locked down NFC for third-party apps
- Expensive, support nightmare: too many platforms, operating systems, browsers,…
- Internal tokens (TPM, secure element) cannot be unplugged

Our solution: Pass2Sign



- Signing key split over device and server, but never reconstructed
  → cannot be compromised by malware
- Server checks password, but stores no offline-attackable information
- Server can block account after suspicious activity
- Server doesn’t see message being signed

Rest of this talk

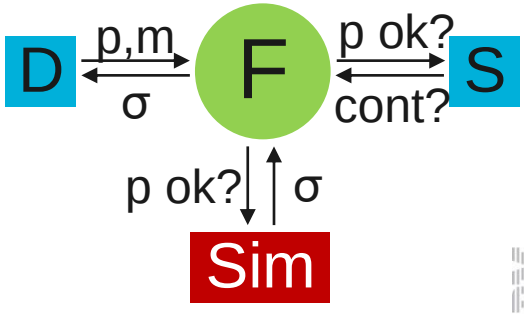
- Related work
- Security model and properties
- Our protocol
- Prototype implementation
- Conclusion

Related work

- Threshold signatures:
  no password that “triggers” signing
  no blindness
- Multi-party computation:
  too inefficient (against adaptive corruptions)
- Server-assisted signatures:
  offline dictionary attacks by corrupted client or server
- S-RSA [MacKenzie-Reiter 2003]
  – corruptions only between signing sessions, not during
  – no blindness
  – active offline dictionary attack by server (but fixable)
  – no universally composable (UC) security

Security model

- Universal composability (UC) to correctly model password distributions, dependencies, typos,…
- Fully adaptive corruptions of device D and server S
  extra-strong: previous inputs not given to adversary
  (needs secure erasures)
- Each signature requires active collaboration
- No offline attacks on passwords, unless device and server both corrupted
- Message blindness: hiding, but no unlinkability
  (adaptive blind signatures in UC notoriously difficult)



Basic protocol

d_D,k

pk = (N,e)
such that
$e \cdot (d_S \cdot d_D) = 1 \bmod \varphi(N)$

d_S, h_p=H(k,p)

$h'_p = H(qid, H(k,p))$

$r \leftarrow \{0,1\}^\tau$, $h_m \leftarrow H(r,m)$

h'_p, h_m →

Check $h'_p = H(qid, h_p)$

← σ_S

$\sigma_S \leftarrow h_m^{d_S} \bmod N$

$\sigma \leftarrow \sigma_S^{d_D} \bmod N$, check $\sigma^e = h_m \bmod N$

Return (σ, r)

Achieving simulatable blindness

d_D, k

d_S, $h_p = H(k,p)$

$h'_p = H(qid, H(k,p))$

$r \leftarrow \{0,1\}^\tau$, $h_m \leftarrow H(r,m)$

Program random oracle "just-in-time" at verification

h'_p, h_m →

Check $h'_p = H(qid, h_p)$

$r' \leftarrow \{0,1\}^t$, $h'_m \leftarrow H(r', h'_m)$

$\sigma_S \leftarrow H'(qid, h'_m)^{d_S} \mod N$

← r', σ_S

$\sigma \leftarrow \sigma_S^{d_D} \mod N$, check $\sigma^e = H'(qid, H(r', h_m)) \mod N$

Return (σ, qid, r, r')

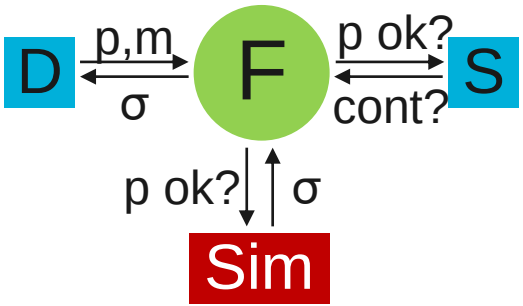
Achieving simulatable blindness

d_D, k

d_S, $h_p = H(k,p)$

$h'_p = H(qid, H(k,p))$

$r \leftarrow \{0,1\}^\tau$, $h_m \leftarrow H(r,m)$



h'_p, h_m

Program random oracle
"just-in-time" at verification

Check $h'_p = H(qid, h_p)$

$r' \leftarrow \{0,1\}^t$, $h'_m \leftarrow H(r', h'_m)$

$\sigma_S \leftarrow H'(qid, h'_m)^{d_S} \mod N$

r', σ_S

$\sigma \leftarrow \sigma_S^{d_D} \mod N$, check $\sigma^e = H'(qid, H(r', h_m)) \mod N$

Return (σ, qid, r, r')

Avoiding offline attacks upon device corruption

d_D,k,epk

d_S, h_p=H(k,p), esk

$h'_p = H(qid, H(k,p))$

$r \leftarrow \{0,1\}^\tau$, $h_m \leftarrow H(r,m)$

Receiver-simulatable non-committing encryption (from any TDP in ROM)

$C \leftarrow Enc(epk, (h'_p,h_m))$

C →

$(h'_p,h_m) \leftarrow Dec(esk, C)$

Check $h'_p = H(qid, h_p)$

$r' \leftarrow \{0,1\}^t$, $h'_m \leftarrow H(r',h'_m)$

$\sigma_S \leftarrow H'(sid, qid, h'_m)^{d_S} \mod N$

← r', σ_S

$\sigma \leftarrow \sigma_S^{d_D} \mod N$, check $\sigma^e = H'(qid, H(r',h_m)) \mod N$

Return (σ, qid, r, r')

Resisting device corruption during signing

d_D,k,epk

d_S, h_p=H(k,p), esk

$h'_p = H(qid, H(k,p))$

$r \leftarrow \{0,1\}^\tau$, $h_m \leftarrow H(r,m)$

$t \leftarrow H(qid,k,h_m)$

Simulator doesn't learn m,p upon device corruption
But device has to verify signature
Send $t \approx MAC_k(m)$ to S and back

$C \leftarrow Enc(epk, (h'_p,h_m,t))$

Store (qid, r), erase m,p,t

C →

$(h'_p,h_m,t) \leftarrow Dec(esk, C)$

Check $h'_p = H(qid, h_p)$

$r' \leftarrow \{0,1\}^t$, $h'_m \leftarrow H(r',h'_m)$

$\sigma_S \leftarrow H'(sid, qid, h'_m)^{d_S} \mod N$

← h_m, t, r', σ_S

Check $t = H(qid,k,h_m)$

$\sigma \leftarrow \sigma_S^{d_D} \mod N$, check $\sigma^e = H'(qid, H(r',h_m)) \mod N$

Return (σ, qid, r, r'), erase everything else

Prototype implementation

- Java 8, no particular optimizations
- Server: laptop, 2.7 GHz, 16 GB RAM, Windows 7
- Device: Nexus 10 tablet, 1.6 GHz, 2 GB RAM, Android 5.1.1

	Setup			Sign		
	1024	2048	4096	1024	2048	4096
Device	648	3335	14343	19	80	483
Server	14	64	388	12	65	456

Median computation times for different key sizes (ms)

Conclusion

Smartcard security*, without the hassle

* as long as not both device and server are corrupted

virtual smart cards

Security reasons for hardware tokens

- Secret key never reconstructed
  → cannot be compromised by malware
- PIN/password protection,
  blocks after too many failed attempts
- Cannot sign when deactivated
- Stolen or lost → revoke certificate

Disadvantages of hardware tokens

- Inconvenient to carry
- External USB reader
- Mobile devices have no USB and/or closed-down NFC
- Support nightmare: too many platforms, operating systems, browsers,…
- Internal tokens cannot be unplugged