Towards Verified C specification for TPM API

Aybek Mukhamedov, Andy Gordon and Mark Ryan
Trusted Platform Module

- Trusted Computing, an initiative by major IT vendors
- Functions: integrity measurement, reporting & protected storage
- Relies on a security chip **TPM**: Root of Trust for Storage and Reporting
- Manual inspection is not feasible (>90 commands), recalls are expensive: need for formal analysis
- Utilize protocol verification tools
Protocol Verification

• A number of successful frameworks: specialized (AVISPA, ProVerif) and general-purpose (FDR, PRISM, Isabelle, mocha, nusmv)

• Abstract models hard to distill, implementations may deviate and revisions may get out of sync
  • hence, hot topic: implementation analysis

• FS2PV: a framework by Gordon et al. (MSRC)
Innovative use of cryptographic constructs often results in subtle (or not so subtle) mistakes. Using standard algorithms in standard ways, or getting expert advice from the crypto board greatly reduces the odds of a problem.

Our goal is a toolkit to verify reference implementations of standardized and custom cryptographic protocols.

```
// a refined type for strings that are genuine requests
type request = s:string [Request(s)]

// a key trusted for authenticating requests
val k: request hkey

let verify message =
  let payload, mac = unpickle message in
  // is this the right cryptographic check?
  let request = hmacVerify k payload mac in request

let server() =
  let socket = Net.listen addr in
  let message = Net.recv socket in
  let request = verify message in
  // can I be sure this request is authentic?
  assert Request(request);
  let response = handle request in
  let msg2 = emitResponse request response in
  Net.send socket msg2
```

Verification Tools for F#
- Statically verify security assertions
- Different techniques, cryptographic models

**CASE STUDIES**
- WS-Security
  - 1750 lines
  - fs2pv [MSRC’06]
- CardSpace
  - 1420 lines
  - fs2pv [MSRC’08]
- TLS 1.0
  - 2940 lines
  - fs2pv, fs2cv [MSR-INRIA’08]
- Multi-party Sessions
  - 2180 lines
  - f7 [MSR-INRIA’08]
TPM API Analysis

Authorization Protocols:

- Important for TPM functionality
- Weak secret off-line attacks by Chen, Ryan (‘08)

Our work:

- Concrete implementation of authorization, encrypted transport session protocols in F#
- Formal verification with FS2PV: attacks, verified our fixes
- F2C: automatic translation into executable C code
Authorization Data

• Most TPM commands & objects require authorization
• Realised via a shared secret authdata
• authdata(20B) is chosen by client app:
  – ideally, it’s a secret + high-entropy data
  – on-line dictionary attack mitigation by TPM
• Ad-hoc analysis by Chen and Ryan (‘08): off-line attack revealing low-entropy authdata
Authorization Protocols

• Object Independent Authorization Protocol (OIAP):
  – can authorize several entities within a session
  – authorization via hmac digest keyed on authdata

• Object Specific Authorization Protocol (OSAP):
  – tied to a single entity
  – authorization via hmac digest keyed on a secret derived from authdata

• Most commands allow either protocol
Object-Independent Pattern (OIAP)

**TPM_OIAP_IN**: tag || paramSize || ordinal

**TPM_OIAP_OUT**: tag || paramSize || returnCode || authHandle || nonceEven

**TPM_OwnerClear_IN**: tag || paramSize || ordinal || authHandle || nonceOdd || continueAuthSession || inAuth

\[ \text{inAuth} = \text{hmacsha1} \text{ authData}(\text{concat1} \ || \ \text{concat2}) \]
\[ \text{with} \]
\[ \text{concat1} = \text{sha1}(\text{ordinal}) \]
\[ \text{concat2} = \text{nonceEven} || \text{nonceOdd} || \text{continueAuthSession} \]

**TPM_OwnerClear_OUT**: tag || paramSize || returnCode || nonceEven’ || continueAuthSession || resAuth

\[ \text{resAuth} = \text{hmacsha1} \text{ authData} (\text{concat1} \ || \ \text{concat2}) \]
\[ \text{with} \]
\[ \text{concat1} = \text{sha1}(\text{returnCode} \ || \ \text{ordinal}) \]
\[ \text{concat2} = \text{nonceEven}' || \text{nonceOdd} || \text{continueAuthSession} \]

\[ \text{authData} \text{ is the weak secret} \]
Object-Specific Pattern (OSAP)

\[\text{TPM\_OSAP\_IN} : \text{tag} \mid\mid \text{paramSize} \mid\mid \text{ordinal} \mid\mid \text{entityType} \mid\mid \text{entityValue} \mid\mid \text{nonceOddOSAP}\]

\[\text{TPM\_OSAP\_OUT} : \text{tag} \mid\mid \text{paramSize} \mid\mid \text{returnCode} \mid\mid \text{authHandle} \mid\mid \text{nonceEven} \mid\mid \text{nonceEvenOSAP}\]

\[\text{TPM\_OwnerClear\_IN} : \text{tag} \mid\mid \text{paramSize} \mid\mid \text{ordinal} \mid\mid \text{authHandle} \mid\mid \text{nonceOdd} \mid\mid \text{continueAuthSession} \mid\mid \text{inAuth}\]

\[\text{inAuth} = \text{hmacsha1 sharedSecret}(\text{concat1} \mid\mid \text{concat2})\]
with \[\text{sharedSecret} = \text{hmacsha1 authData (nonceOddOSAP} \mid\mid \text{nonceEvenOSAP)}\]
\[\text{concat1} = \text{sha1(ordinal)}\]
\[\text{concat2} = \text{nonceEven} \mid\mid \text{nonceOdd} \mid\mid \text{continueAuthSession}\]

\[\text{TPM\_OwnerClear\_OUT} : \text{tag} \mid\mid \text{paramSize} \mid\mid \text{returnCode} \mid\mid \text{nonceEven'} \mid\mid \text{continueAuthSession} \mid\mid \text{resAuth}\]

\[\text{resAuth} = \text{hmacsha1 sharedSecret (concat1} \mid\mid \text{concat2})\]
with \[\text{sharedSecret} = \text{hmacsha1 authData (nonceOddOSAP} \mid\mid \text{nonceEvenOSAP)}\]
\[\text{concat1} = \text{sha1(returnCode} \mid\mid \text{ordinal})\]
\[\text{concat2} = \text{nonceEven'} \mid\mid \text{nonceOdd} \mid\mid \text{continueAuthSession}\]

\text{authData is the weak secret}
Encrypted Transport Protection

Start auth session, load secret encryption key

**TPM_EstablishTransport** _IN_: encHandle || encr. secret || authHandle || keyAuth

**TPM_ExecuteTransport** _IN_: wrappedCmd, transHandle || transNonceOdd || transAuth

**TPM_EstablishTransport** _OUT_: transHandle || transnonceEven || resAuth

**TPM_ExecuteTransport** _OUT_: wrappedRsp, transHandle || transNonceEven’ || transAuth

wrappedCmd = TAGw || LENw || ORDw || HANDLESw || DATAw || AUTH1w

wrappedRsp = TAGw || LENw || RCw || HANDLESw || DATAw || AUTH1w

tranEncKey = transNonceEven’ || transNonceOdd || “out” || secret
Concrete specification in F#

• We implemented authorization and transport protection protocols

Snippet of fTPM.fs (TPM code)

```fsharp
let TPM_OSAP (input:TPM_OSAP_IN) : TPM_OSAP_OUT =
if (input.tag_osapIn = TPM_TAG_RQU_COMMAND) then
  if (input.ordinal_osapIn = TPM_ORD_OSAP) then
    let nonceEven : TPM_NONCE = mkNonceEven() in
    let nonceEvenOSAP : TPM_NONCE = mkNonce() in
    let xNonceOddOSAP : TPM_NONCE =
      input.nonceOddOSAP_osapIn
    let hmac_data : BYTES = dconcat nonceEvenOSAP
     xNonceOddOSAP in
    let handle : TPM_AUTHHANDLE = allocHandle() in
    let entityType : TPM_ENTITY_TYPE = input.entityType_osapIn
    if (entityType=TPM_ET_OWNER) then begin [...]]
```

Data defs

```fsharp
type TPM_NONCE = {
  nonce: BYTE20 }

let TPM_ORD_OSAP = (UINT32 0x0000000B)
let TPM_ORD_OIAP = (UINT32 0x0000000A)

type TPM_OSAP_IN = {
  tag_osapIn : TPM_TAG;
  ordinal_osapIn : UINT32;
  entityType_osapIn : TPM_ENTITY_TYPE;
  entityValue_osapIn : UINT32;
  nonceOddOSAP_osapIn : TPM_NONCE
}```
Verification with FS2PV

- Symbolic libraries: **Crypto, Data, Net, Db**
  
  ```fsharp
  type bytes =
  | SymEncrypt of bytes*bytes
  | Hmacsha1 of bytes*bytes
  | Num32 of unit32
  | Concat of bytes*bytes

  let sym_encrypt key text = SymEncrypt(key,text)
  let sym_decrypt key msg = match msg with
  | SymEncrypt(key',text) when key = key' -> text
  | _ -> failwith "sym_decrypt failed"
  ```

- Symbolic execution: sanity check, debugger

- Active attacker:
  - crypto, data manipulation, network control defined via F# interface files

- FS2PV generates ProVerif model, which is verified against any number of TPM, Client runs & arbitrary attacker
Verification Results

Found reported attacks, issues with transport protection and proved correctness of our fixes

<table>
<thead>
<tr>
<th></th>
<th>LoC in F# (incl. data, type defs)</th>
<th>LoC in C (incl. data, type defs)</th>
<th>FS2PV analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSAP session</td>
<td>265</td>
<td>275</td>
<td>found attack</td>
</tr>
<tr>
<td>OIAP session</td>
<td>250</td>
<td>260</td>
<td>found attack</td>
</tr>
<tr>
<td>Encrypted Transport Session OIAP</td>
<td>780</td>
<td>660</td>
<td>found attack &amp; proved fix when all commands in OIAP session encrypt nonceEven</td>
</tr>
<tr>
<td>Encrypted Transport Session OSAP</td>
<td>800</td>
<td>680</td>
<td>found attack &amp; proved fix when TPM_OSAP response is encrypted</td>
</tr>
</tbody>
</table>
F2C: executable C from F# code

- F# is not widely adopted and relies on .NET
- **F2C**: translates verified F# code of the TPM into executable C implementation:
  - TPM commands, data structures
- Built on top of FS2PV library
- Implemented sample client and crypto functions in C++ for authorization sessions
Client --> TPM: TPM_OSAP msg1
where
msg1.tag=193,
msg1.pramSize=36,
msg1.ordinal=11,
msg1.entityType=2,
msg1.entityValue=0,
msg1.nonceOddOSAP=8b9e537d48cc1502a1d15d842faa0473efdb06

TPM --> Client: TPM_OSAP msg2,
where
msg2.tag=196,
msg2.pramSize=54,
msg2.returnCode=0,
msg2.authHandle=0x0,
msg2.nonceEven=8519b61d32c1114e58ef830708c0f3d5eb4c,
msg2.nonceOddOSAP=1cd26b89129929c08239944a458156a62bdfed8

Event: Client_Issues_OSAP\OwnerAuthSecret,msg1.nonceOddOSAP,msg2.nonceEvenOSAP,msg3.nonceOdd,msg2.nonceEven

Client --> TPM: TPM_OwnerClear msg3
where
msg3.tag=194,
msg3.pramSize=58,
msg3.ordinal=91,
msg3.authHandle=0x0,
msg3.nonceOdd=3f2e1662ba0b1f4f0e391d7728d626600452cbb0,
msg3.continueAuthSession=0,
msg3.ownAuth= hmacsha1 sharedSecret <concat1 || concat2>
with
sharedSecret= hmacsha1 ownerAuthSecret <msg1.nonceOddOSAP || msg2.nonceEvenOSAP>
concat1= sha1(msg3.ordinal)
concat2= msg2.nonceEven || msg3.nonceOdd || msg3.continueAuthSession

Event: TPM_Accepts_OwnerClear\ownerAuthSecret,msg1.nonceOddOSAP,msg2.nonceEvenOSAP,msg3.nonceOdd,msg2.nonceEven


TPM --> Client: TPM_OwnerClear msg4,
where
msg4.tag=196,
msg4.pramSize=54,
msg4.returnCode=0,
msg4.nonceEven=293ba35934e30a5e9b3e22441e1e85c4599693,
msg4.continueAuthSession=0,
msg4.ownAuth= hmacsha1 sharedSecret <concat1 || concat2>
with
sharedSecret= hmacsha1 ownerAuthSecret <msg1.nonceOddOSAP || msg2.nonceEvenOSAP>
concat1= sha1(msg4.returnCode || msg4.ordinal)
concat2= msg4.nonceEven || msg3.nonceOdd || msg4.continueAuthSession

Event: Client_Accepts_OwnerClearResponse\ownerAuthSecret,msg1.nonceOddOSAP,msg2.nonceEvenOSAP,msg4.returnCode,msg3.nonceOdd,msg4.nonceEven

Press any key to exit ...
Conclusions

• Verified concrete implementation of TPM API fragment
• Captured attacks and verified fixes
• F2C: C code generator from a stylized F#

Future work:
• Many more commands to cover, interoperability
• Improve verification engine: mutable states
• Generate F# code from C spec

Verified reference implementation of the whole TPM seems plausible