

Typing Correspondence Assertions for Cryptographic Protocols (Work in Progress)

Andrew D. Gordon, Microsoft Research
Alan Jeffrey, DePaul University

Overview

- Introduction
- Example
- Traces and correspondence assertions
- Type system
- Conjectures (hopefully to become theorems soon!)
- Conclusions

Background

Ensuring safety and security properties of cryptographic protocols is hard! There is a large literature, including:

- *Modal logics* such as BAN logic, checked by hand (Burrows, Abadi and Needham).
- *Process equivalence* such as testing equivalence, checked by hand (Abadi and Gordon) and (Schneider)
- *Model checking* can automate either of these techniques to verify finite-state systems (Lowe) and (Clarke, Jha and Marrero).

Contributions of this work

The new contributions of this work are:

- Instrumenting spi-calculus programs with correspondence assertions to validate security properties.
- Using a type/effect system to ensure that assertions never fail.
- A novel type for nonces, which records the trust information a nonce carries.

Using this technique on examples seems to be much easier than previous spi-calculus techniques.

Insecure example

Informal description:

$$Sender \rightarrow Receiver : \{M\}_K$$

Formal description of principals:

$$\begin{array}{ll} Sender(net, K) \triangleq & Receiver(net, K) \triangleq \\ \text{repeat} & \text{repeat} \\ \text{new}(M); & \text{inp } net(C); \\ \text{out } net(\{M\}_K) & \text{decrypt } C \text{ is } \{M\}_K; \end{array}$$

Formal description of system:

$$\begin{array}{l} System(net) \triangleq \\ \text{new}(K); (Sender(net, K) \mid Receiver(net, K)) \end{array}$$

Insecure example cont.

Informal description of safety property:

Sender \rightarrow *Receiver* : $\{M\}_K$

Receiver believes *Sender* sent M

Instrument protocol with assertions:

Sender(*net*, K) \triangleq

repeat

new(M);

allow (*Sender* sent M);

out net($\{M\}_K$)

Receiver(*net*, K) \triangleq

repeat

inp net(C);

decrypt C is $\{M\}_K$;

assert (*Sender* sent M)

Then check the safety property:

Every *assert* M has a matching *allow* M

This is an example of a *correspondence assertion* (Woo and Lam).

Traces

To formalize correspondence assertions, we record the *traces* of a system, for example:

- One trace of *assert M* is *assert M*.
- One trace of *allow M; assert M* is *allow M, assert M*.
- One trace of *new(x); allow x; assert x* is *gen x, allow x, assert x*.
- One trace of *decrypt {M}_K is {x}_K; assert x* is *assert M*.
- One trace of *System(net)* is *gen M, allow (Sender sent M), assert (Sender sent M)*.

Safety

Define:

- A trace s is a *correspondence* iff for every M , the number of *assert* M s in s is less than or equal to the number of *allow* M s in s .
- A process P is *safe* iff every trace of P is a correspondence.

System(net) is safe, e.g. the following trace is a correspondence:

gen M , *allow* (Sender sent M), *assert* (Sender sent M)

Robust safety

Define:

- A process P is *robustly safe* iff for every assert-free opponent process O , the process $P \mid O$ is safe.

$System(net)$ is *not* robustly safe, due to replay attacks:

$$Attacker(net) \triangleq \\ inp\ net(C); out\ net(C); out\ net(C)$$

One trace of $System(net) \mid Attacker(net)$ is:

$gen\ M, allow\ (Sender\ sent\ M),$
 $assert\ (Sender\ sent\ M), assert\ (Sender\ sent\ M)$

This trace is not a correspondence, so $System(net)$ is not robustly safe.

Fixed example

Informal description:

Receiver \rightarrow *Sender* : N

Sender \rightarrow *Receiver* : $\{M, N\}_K$

Formal description:

Sender(net, K) \triangleq

repeat

inp $net(N)$;

new(M);

allow (Sender sent M);

out $net(\{M, N\}_K)$

Receiver(net, K) \triangleq

repeat

new(N);

out $net(N)$;

inp $net(C)$;

decrypt C is $\{M, N'\}_K$;

check N' is N ;

assert (Sender sent M)

Types

- $Ch(T)$ is a channel type for communicating T .
- $Nonce [assert M]$ is a nonce type which allows $assert M$ safely.
- $Key(T)$ a symmetric key type for encrypting and decrypting plaintext T and cyphertext Un .
- Un is type of untrusted data.

for example:

$$Network \triangleq Ch(Un)$$

$$MyNonce(M) \triangleq Nonce [assert (Sender\ sent\ M)]$$

$$MyKey \triangleq Key(M : Msg, N : MyNonce(M))$$

Type theorists may note the use of dependent records.

Fixed example cont.

Type-safe receiver:

$$\begin{aligned} & \text{Receiver}(\text{net}:\text{Network}, K:\text{MyKey}) \triangleq \\ & \quad \text{repeat} \\ & \quad \quad \text{new}(N:\text{Un}); \\ & \quad \quad \text{out net}(N); \\ & \quad \quad \text{inp net}(C:\text{Un}); \\ & \quad \quad \text{decrypt } C \text{ is } \{M:\text{Msg}, N':\text{MyNonce}(M)\}_K; \\ & \quad \quad \text{check } N' \text{ is } N; \\ & \quad \quad \text{assert}(\text{Sender sent } M) \end{aligned}$$

where:

$$\begin{aligned} \text{Network} & \triangleq \text{Ch}(\text{Un}) \\ \text{MyNonce}(M) & \triangleq \text{Nonce}[\text{assert}(\text{Sender sent } M)] \\ \text{MyKey} & \triangleq \text{Key}(M : \text{Msg}, N : \text{MyNonce}(M)) \end{aligned}$$

Fixed example cont.

Almost type-safe sender:

$$\begin{aligned} \text{Sender}(\text{net}:\text{Network}, K:\text{MyKey}) &\triangleq \\ \text{repeat} & \\ \text{inp } \text{net}(N:\text{Un}); & \\ \text{new}(M:\text{Msg}); & \\ \text{allow } (\text{Sender sent } M); & \\ \text{out } \text{net}(\{M, N\}_K) & \end{aligned}$$

where:

$$\begin{aligned} \text{Network} &\triangleq \text{Ch } (\text{Un}) \\ \text{MyNonce } (M) &\triangleq \text{Nonce } [\text{assert } (\text{Sender sent } M)] \\ \text{MyKey} &\triangleq \text{Key } (M : \text{Msg}, N : \text{MyNonce } (M)) \end{aligned}$$

but $\{M, N\}_K$ doesn't typecheck, because $N : \text{Un}$.

Fixed example cont.

We add type-casting into the language:

$$\begin{aligned} \text{Sender}(\text{net}:\text{Network}, K:\text{MyKey}) &\triangleq \\ \text{repeat} & \\ \text{inp } \text{net}(N:\text{Un}); & \\ \text{new}(M:\text{Msg}); & \\ \text{allow } (\text{Sender sent } M); & \\ \text{cast } N \text{ is } (N':\text{MyNonce } (M)); & \\ \text{out } \text{net}(\{M, N'\}_K) & \end{aligned}$$

where:

$$\begin{aligned} \text{Network} &\triangleq \text{Ch } (\text{Un}) \\ \text{MyNonce } (M) &\triangleq \text{Nonce } [\text{assert } (\text{Sender sent } M)] \\ \text{MyKey} &\triangleq \text{Key } (M : \text{Msg}, N : \text{MyNonce } (M)) \end{aligned}$$

At run-time $\text{cast } N \text{ is } (x:T); P$ becomes $P\{x \leftarrow N\}$.

Effect system

The type system calculates the *effect* of a process P :

- if P has effect $[\textit{assert } M]$
then P may assert M without allowing it first.
- if P has effect $[\textit{nonce } N]$
then P may use N as a nonce.

For example:

- $\textit{assert } M; \textit{assert } M'$ has effect $[\textit{assert } M, \textit{assert } M']$
- $\textit{allow } M; \textit{assert } M; \textit{assert } M'$ has effect $[\textit{assert } M']$
- $\textit{check } N' \textit{ is } N; \textit{assert } M$ has effect $[\textit{nonce } N]$
- $\textit{new}(N:Un); \dots \textit{check } N' \textit{ is } N; \textit{assert } M$ has effect $[]$

where $N : Un$ and $N' : Nonce [\textit{assert } M]$

Effect system

The interesting rules for calculating the effect of P are:

- $assert M$ has effect $[assert M]$.
- If P has effect $[es, assert M]$
then $allow M; P$ has effect $[es]$
- If P has effect $[es]$
then $cast M is (x:Nonce [fs]); P$ has effect $[es, fs]$
- If P has effect $[es, fs]$ and $N' : Nonce [fs]$
then $check N' is N; P$ has effect $[es, nonce N]$.
- If P has effect $[es, nonce x]$
then $new(x:T); P$ has effect $[es]$.

Fixed example cont.

The effect of $Sender(net, K)$ is calculated:

$$\begin{aligned} &Sender(net:Network, K:MyKey) \triangleq \\ &\quad \textit{repeat} \\ &\quad \quad \textit{inp } net(N:Un); \\ &\quad \quad \textit{new}(M:Msg); \\ &\quad \quad \textit{allow} (\textit{Sender sent } M); \\ &\quad \quad \textit{cast } N \textit{ is } (N':MyNonce (M)); \\ &\quad \quad \textit{out } net(\{M, N'\}_K) \} [] \end{aligned} \left. \vphantom{\begin{aligned} &\quad \quad \textit{allow} (\textit{Sender sent } M); \\ &\quad \quad \textit{cast } N \textit{ is } (N':MyNonce (M)); \\ &\quad \quad \textit{out } net(\{M, N'\}_K) \} []} \right\} [assert \dots] \left. \vphantom{\left. \vphantom{\begin{aligned} &\quad \quad \textit{allow} (\textit{Sender sent } M); \\ &\quad \quad \textit{cast } N \textit{ is } (N':MyNonce (M)); \\ &\quad \quad \textit{out } net(\{M, N'\}_K) \} []} \right\} [assert \dots]} \right\} [] \end{aligned}$$

where:

$$\begin{aligned} Network &\triangleq Ch (Un) \\ MyNonce (M) &\triangleq Nonce [\textit{Sender sent } M] \\ MyKey &\triangleq Key (M : Msg, N : MyNonce (M)) \end{aligned}$$

Fixed example cont.

$Receiver(net:Network, K:MyKey) \triangleq$
repeat
 new($N:Un$);
 out net(N);
 inp net($C:Un$);
 decrypt C is $\{M:Msg, N':MyNonce(M)\}_K$;
 check N' is N ;
 assert (Sender sent M) } [*assert ...*] } [*nonce N*]

where:

$Network \triangleq Ch(Un)$

$MyNonce(M) \triangleq Nonce[Sender\ sent\ M]$

$MyKey \triangleq Key(M : Msg, N : MyNonce(M))$

Conjectures

We are currently working on proving:

- If P has empty effect then P is safe.
- If P has empty effect, and P only has free variables of type $Ch(U_n)$ then P is robustly safe.

Most of the work is in place, the proof is a subject reduction result.

Contributions

We have annotated the spi-calculus with correspondence assertions.

We have provided a type/effect system for the annotated processes, which guarantees that assertions will never fail.

Verifying a 7-message protocol took Abadi and Gordon 2-3 person months with existing spi methods. It took Gordon an afternoon using ours.

To Do

Prove the conjectures!

Examples (how much of Kerberos can we verify?).

Deal with partially trusted attackers.