Cryptyc Tutorial

Scott A Crosby

September 5, 2002

Congratulations! You're lucky!

You get to use Cryptyc to prove your protocols correct..
This means you avoid a miserable mess of proofs.

What is Cryptyc?

The Cryptyc system is a Cryptographic Protocol Type Checker. Unlike typecheckers you may be familiar with, which check for simple data errors, such as dereferencing an integer or treating a pointer as a boolean. It can also statically check for security violations such as secrecy or authenticity errors.

If it is happy with your protocol (and you don't cheat), you can have some trust in the correctness of your design.

If it isn't happy... Well, who knows.

Cryptyc Language

The input file is a language of processes.. You can create new datums, you can initiate connections, you can receive and check results.

```
initiate Receiver at Bob is (socket : Socket);
output socket is (msg);
output socket is ({ msg, nonce }SKey);
new (msg : Public);
input socket is ({ msg : Payload, nonce' : Challenge }SKey);
check nonce is nonce';
```


Processes are different entities that all run concurrently. They only rendezvous during communication.

A 'Secure' Protocol

Here's a protocol. We have Alice and Bob. Bob wants to be able to safely assert that by the time Bob terminates, Bob knows that Alice sent msg.

We've got a formal definition here.. Of an insecure protocol. Cryptyc will accept it because we haven't defined what security properties we wish it to have.

A 'Secure' Protocol (cont'd)

```
client Sender at Alice is {
  initiate Receiver at Bob is (socket : Socket);
  new (msg : Public);
  output socket is (msg);
}

server Receiver at Bob is (socket : Socket) {
  input socket is (msg : Public);
}
```

A Secure Protocol

A secure variant. We encrypt the message and use a nonce handshake to prevent replay attacks.

```
client Sender at Alice is {
  establish Receiver at Bob is (socket : Socket);
  input socket is (nonce : Challenge);
  new (msg : Payload);
  output socket is ({ msg, nonce }SKey);
}
```

A Secure Protocol (cont'd)

```
server Receiver at Bob is (socket : Socket) {
  new (nonce : Challenge);
  output socket is (nonce);
  input socket is ({ msg : Payload, nonce' : Challenge }SKey);
  check nonce is nonce';
}
```

This variant is secure, but we've not actually stated any security properties that we want.

But what is this?

We must include ASSERTIONS into a protocol, Like:

```
client Sender at Alice is {
  initiate Receiver at Bob is (socket : Socket);
  new (msg : Public);
  begin (Alice sent msg to Bob);
  output socket is (msg);
}
server Receiver at Bob is (socket : Socket) {
  input socket is (msg : Public);
  end (Alice sent msg to Bob);
}
```

∞

But what is this? (cont'd)

```
server Receiver at Bob is (socket : Socket) {
  input socket is (msg : Public);
  end (Alice sent msg to Bob);
}
```

Now, Cryptyc will call a protocol secure if for every `endX`, a corresponding `beginX` must be invoked.

Now that we've told Cryptyc what authenticity properties we want, it'll say (correctly) here that we don't have them.

The Secure Protocol with assertions

```
client Sender at Alice is {
  establish Receiver at Bob is (socket : Socket);
  input socket is (nonce : Challenge);
  new (msg : Payload);
  begin (Sender sent msg);
  output socket is ({ msg, nonce }SKey);
}
server Receiver at Bob is (socket : Socket) {
  new (nonce : Challenge);
  output socket is (nonce);
  input socket is ({ msg : Payload, nonce' : Challenge }SKey);
  check nonce is nonce';
  end (Sender sent msg);
}
```

Lesson?

The lesson. If Cryptyc complains about mismatched effects, remove the assertions. The protocol that says nothing can't be wrong, can it?

Type annotations

Available types:

- `Public`
- `Private`
- `Key(T)`
- `Struct (foo: T1, bar: T2, baz: T3)`
- `Union (tag1: T1, tag2: T2, tag3: T3)`
- `Nonce (end (M))`

Type annotations. (cont'd)

We have:

```
type MyNonce (msg : Payload) = Nonce (end (Sender sent msg));
type MyMsg = Struct (msg : Payload, nonce : MyNonce(msg));
type MyKey = Key (MyMsg);
```

Where the private key has type `MyKey`:

```
private SKey : MyKey;
```

What is this payload in the Nonce? 'Sender sent msg'?

Type annotations. (cont'd)

```
client Sender at Alice is {
  establish Receiver at Bob is (socket : Socket);
  input socket is (nonce : Challenge);
  new (msg : Payload);
  begin (Sender sent msg);
  cast nonce is (nonce' : MyNonce (msg));
  output socket is ({ msg, nonce' }SKey);
}
```

Type annotations. (cont'd)

```
server Receiver at Bob is (socket : Socket) {
  new (nonce : Challenge);
  output socket is (nonce);
  input socket is ({ msg : Payload, nonce' : MyNonce (msg)
                   }SKey);
  check nonce is nonce';
  end (Sender sent msg);
}
```

Important points

Effects are sent between processes by being attached to nonces:

- Note definition of `MyNonce`
- Note use of `check` to check the nonce.
- Note use of `cast` to satisfy a nonce, to convert `Untyped` to `:MyNonce(msg) = Nonce (end (Sender sent msg))`
- Cryptyc won't let you check a nonce more than once.

Woo and Lam (modified)

```
*    A -> B : A
*    B -> A : Nb
*    A -> B : { msg3(B, Nb) }Kas
*    B -> S : A, B, { msg3(B, Nb) }Kas
*    S -> B : { msg5(A, Nb) }Kbs
```

Woo and Lam (modified) (cont'd)

```
type WLNonce (a : Host, b : Host) =
  Nonce (end (a authenticates to b));

type WLMsg3 (a : Host) =
  Struct (b : Host, nonce : WLNonce (a, b));
type WLMsg5 (b : Host) =
  Struct (a : Host, nonce : WLNonce (a, b));
type WLMsg (h : Host) =
  Union (msg3 : WLMsg3 (h), msg5 : WLMsg5 (h));

type WLKey (h : Host) =
  Key (WLMsg (h));

type WLKDCMsg = Struct (h : Host, keyH : WLKey (h));
```

Woo and Lam (modified) (cont'd)

The keys:

```
type WLKDCKey = Key (WLKDCMsg);

private keyA : WLKey (Alice);
private keyB : WLKey (Bob);
private keyKDC : WLKDCKey;
```

Woo and Lam (modified) (cont'd)

```
client Initiator at Alice is {
  begin (Alice authenticates to Bob);
  establish Responder at Bob is (socket : Socket);
  output socket is (Alice);
  input socket is (nonce : Challenge);
  cast nonce is (nonce' : WLNonce (Alice, Bob));
  output socket is ({ msg3 (Bob, nonce') }keyA);

}
```

Woo and Lam (modified) (cont'd)

```
server Responder at Bob is (socketA : Socket) {
  input socketA is (a : Host);
  new (nonce : Challenge);
  output socketA is (nonce);
  input socketA is (cText : CText);
  establish Intermediary at Sam is (socketS : Socket);
  output socketS is (a, Bob, cText);
  input socketS is ({ msg5 (a, nonce' : WLNonce (a, Bob)) }keyB);
  check nonce is nonce';
  end (a authenticates to Bob);

}
```

Woo and Lam (modified) (cont'd)

```
server Intermediary at Sam is (socketB : Socket) {
  input socketB is (a : Host, b : Host, cText : CText);
  establish KDC at KDCHost is (socketKDC : Socket);
  output socketKDC is (a);
  input socketKDC is ({ a, keyA : WLKey (a) }keyKDC);
  output socketKDC is (b);
  input socketKDC is ({ b, keyB : WLKey (b) }keyKDC);
  decrypt cText is ({ msg3 (b, nonce' : WLNonce (a, b)) }keyA);
  output socketB is ({ msg5 (a, nonce') }keyB);

}
```