# PROTOCOL VERIFICATION IN THE COMPUTATIONAL MODEL

(using EasyCrypt)

#### NAXOS

$$A \qquad \mathcal{B}$$

$$esk_{\mathcal{A}} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda} \qquad X = g^{H_{1}(esk_{\mathcal{A}},sk_{\mathcal{A}})} \qquad esk_{\mathcal{B}} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda}$$

$$Y = g^{H_{1}(esk_{\mathcal{B}},sk_{\mathcal{B}})} \qquad esk_{\mathcal{B}} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda}$$

$$\mathcal{B}: K \leftarrow H_{2}(pk_{\mathcal{A}}^{H_{1}(esk_{\mathcal{B}},sk_{\mathcal{B}})}, X^{sk_{\mathcal{B}}}, X^{H_{1}(esk_{\mathcal{B}},sk_{\mathcal{B}})}, A, \mathcal{B})$$

$$A: K \leftarrow H_{2}(Y^{sk_{\mathcal{A}}}, pk_{\mathcal{B}}^{H_{1}(esk_{\mathcal{A}},sk_{\mathcal{A}})}, Y^{H_{1}(esk_{\mathcal{A}},sk_{\mathcal{A}})}, A, \mathcal{B})$$

#### AKE PROTOCOLS

$$A \qquad \mathcal{B}$$

$$esk_{\mathcal{A}} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda} \qquad X = g^{H_{1}(esk_{\mathcal{A}},ek_{\mathcal{A}})} \qquad esk_{\mathcal{B}} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda}$$

$$Y = g^{H_{1}(esk_{\mathcal{B}},ek_{\mathcal{B}})} \qquad esk_{\mathcal{B}} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda}$$

$$\mathcal{B}: K \leftarrow H_{2}(pk_{\mathcal{A}}^{H_{1}(esk_{\mathcal{B}},sk_{\mathcal{B}})}, X^{sk_{\mathcal{B}}}, X^{H_{1}(esk_{\mathcal{B}},sk_{\mathcal{B}})}, \mathcal{A}, \mathcal{B})$$

$$\mathcal{A}: K \leftarrow H_{2}(Y^{sk_{\mathcal{A}}}, pk_{\mathcal{B}}^{H_{1}(esk_{\mathcal{A}},sk_{\mathcal{A}})}, Y^{H_{1}(esk_{\mathcal{A}},sk_{\mathcal{A}})}, \mathcal{A}, \mathcal{B})$$

#### AKE PROTOCOLS

```
A
esk_{\mathcal{A}} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda} \qquad X = g^{H} \text{ out(eskA,skA)}
Y = g^{H} \text{ in(eskB,skB)} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda}
\mathcal{B}: \quad \text{H(gen\_session\_string(...))} \stackrel{\text{csk}_{\mathcal{A}}, sk_{\mathcal{A}}}{\rightarrow}, \mathcal{A}, \mathcal{B})
A: \quad \text{H(gen\_session\_string(...))} \stackrel{k_{\mathcal{A}}, sk_{\mathcal{A}}}{\rightarrow}, \mathcal{A}, \mathcal{B})
```

#### AKE PROTOCOLS

$$A$$

$$esk_{\mathcal{A}} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda} \qquad X = g^{H} \text{ out(eskA,skA)}$$

$$Y = g^{H} \text{ in(eskB,skB)} \stackrel{\text{esk}_{\mathcal{B}}}{\leftarrow} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda}$$

$$\mathcal{B}: \quad \text{H(gen\_session\_string(...))} \stackrel{\text{h.a.sk}_{\mathcal{A}}}{\rightarrow} \mathcal{A}: \mathcal{B})$$

how much of the proof can we get away with without instantiating?

#### ORACLES

- KG() : public\_key
- Init(A : part, B: part) : msg option
- Respond(B : part, A : part, X : msg) : msg option
- Complete(A:part, B:part, X:msg, Y:msg): unit
- Hash(str : session\_string) : session\_key
- Corrupt(A : part) : private\_key option
- EphKeyReveal(A : part, X : msg) : eph\_key option
- KeyReveal(s : session\_id) : session\_key
- Test(s : session\_id) : session\_key

# ORACLES (HASH, 4 LOC)

```
fun H(lam:session_string) : session_key = {
  var h : session_key = gen_session_key(0);
  if (!in_dom(lam, LH)) { LH[lam] = h; }
  return LH[lam];
}
```

# ORACLES (INIT, ~10 LOC)

```
fun Init(A:part, B:part) : message option = {
 var x : eph_key = gen_eph_key(0);
 var a : secret_key;
 var X : message option = None;
 if (in_dom(A, skey) && in_dom(B, skey)){
   a = skey[A];
  X = Some(out\_noclash(a, x));
  if (!in_dom((proj(X),A), seed)) {
   incomplete_sessions[(A,proj(X))] = (B,false);
   seed[(proj(X),A)] = x;
   }
   else
   X = None;
  return X;}
```

# ORACLES (KEYREVEAL, ~100 LOC)

```
fun KeyReveal(s : session_id) : session_key = {
 var A:part = fstpart(s);
 var B:part = sndpart(s);
 var X:message = fstmsg(s);
 var Y:message = sndmsg(s);
 var x : eph_{key} = seed[(X,A)];
 var B' : part = dummy_part;
 var Y' : message = dummy_msg;
 var eph_flagA : bool = false;
 var eph_flagB : bool = false;
 var A' : part = dummy_part;
 var X' : message = dummy_msq;
 var sstr : session_string = dummy_string;
 var ssskey : session_key = dummy_session_key;
 var matchb : bool = false;
 var h : session_key = dummy_session_key;
 var sidA, sidB : session_id;
 h = gen_session_key(0);
 if (in_dom((A,X), complete_sessions))
 { (* (A,_,X,_) is completed*)
   B' = session_part(complete_sessions[(A,X)]);
   Y' = session_msg(complete_sessions[(A,X)]);
   sidA = mk\_sid(A, B', X, Y');
```

```
eph_flagA = session_eph_flag(complete_sessions[(A,X)]);
     if (B = B' \&\& Y = Y')
     \{(* B = B' / Y = Y'*)\}
     if (!in_dom((B,Y), complete_sessions))
      {(*B,_,Y,_ not complete*)
      if (! in_dom(sidA, tested_session))
       {(*Fresh*)
         complete_sessions[(A,X)] = mk_session_descr(B',Y',eph_flagA,true);
         sstr = gen_session_string_sid(sidA, skey, seed);
             (*sstr = gen_session_string(skey[A], x, B,Y);*)
             ssskey = iH(sstr, h);
      }
       else
       { (*not fresh*)
         ssskey = dummy_session_key;
else
{ (* B,_,Y_ complete *)
       A' = session_part(complete_sessions[(B,Y)]);
      X' = session_msg(complete_sessions[(B,Y)]);
       sidB = mk\_sid(B, A', Y, X');
       matchb = session_match( mk_sid (A, B', X,Y'),mk_sid(B,A',Y,X'));
```

#### MAIN

```
fun Main () : bool = {
var b' : bool;
var tt : unit;
 complete_sessions = empty_map; incomplete_sessions = empty_map;
 corrupt = empty_map; pkey = empty_map;
 skey = empty_map; LH = empty_map;
LHT = empty_map; seed = empty_map;
tested_session = empty_map; G = empty_map;
b = \{0,1\};
b' = A();
return (b = b');
```

Gamel

Gamel

Return random in keyreveal, save queried sessions in G

Gamel

Return random in keyreveal, save queried sessions in G

Game2

Split LH into two

Gamel

Return random in keyreveal, save queried sessions in G

Game2

Calit III into two

Split LH into two

Isolate Bad event: a clash between keyreveal and test or "guessing" a session string

Game4

Gamel

Return random in keyreveal, save queried sessions in G

Game2

Split LH into two

Isolate Bad event: a clash between keyreveal and test or "guessing" a session string

Game4

Game3

Swap b after the call to the adversary: test does not depend on b anymore.

Gamel Game2 Game3 Return random in keyreveal, save queried sessions in G Split LH into two Isolate Bad event: a clash between keyreveal and test or "guessing" a session string Game4 Swap b after the call to the adversary: test does not depend on b anymore. Game5 Probability of wining is 1/2

Gamel

Return random in keyreveal, save queried sessions in G

Game2

Split LH into two

Isolate Bad event: a clash between keyreveal and test or "guessing" a session string

Game4

Game3

Swap b after the call to the adversary: test does not depend on b anymore.

Gamel Game2 Return random in keyreveal, save queried sessions in G Split LH into two session string Game6 Bound clash: reduction step on b anymore.

Game3

Isolate Bad event: a clash between keyreveal and test or "guessing" a

Game4

Swap b after the call to the adversary: test does not depend

Gamel Game3 Game2 Return random in keyreveal, save queried sessions in G Split LH into two Isolate Bad event: a clash between keyreveal and test or "guessing" a add: get rid of: hash, same\_session\_string keyreveal and test and eqS Game4 Game6 Bound clash: reduction step Swap b after the call to the adversary: test does not depend on b anymore. Game5

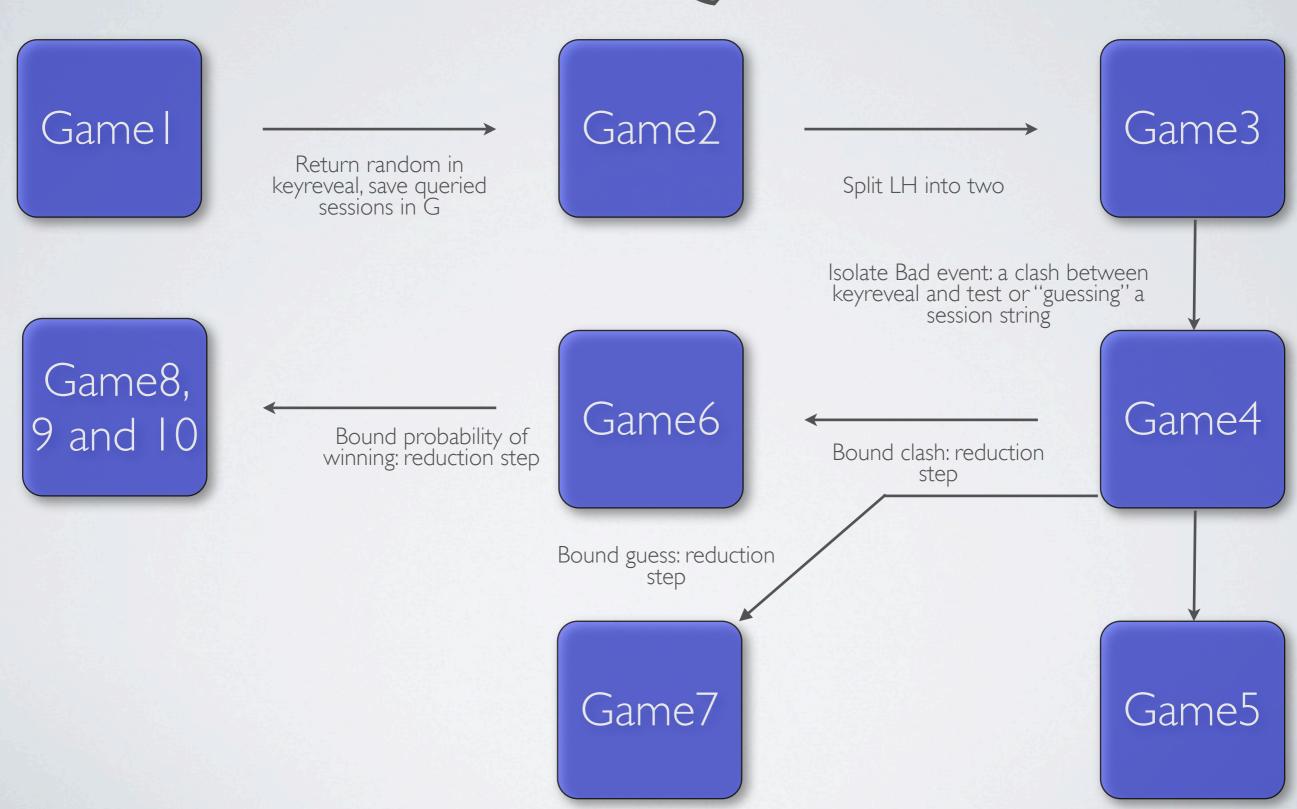
Gamel Game2 Return random in keyreveal, save queried sessions in G Split LH into two Isolate Bad event: a clash between keyreveal and test or "guessing" a session string Game6 Bound clash: reduction step Swap b after the call to the adversary: test does not depend on b anymore.

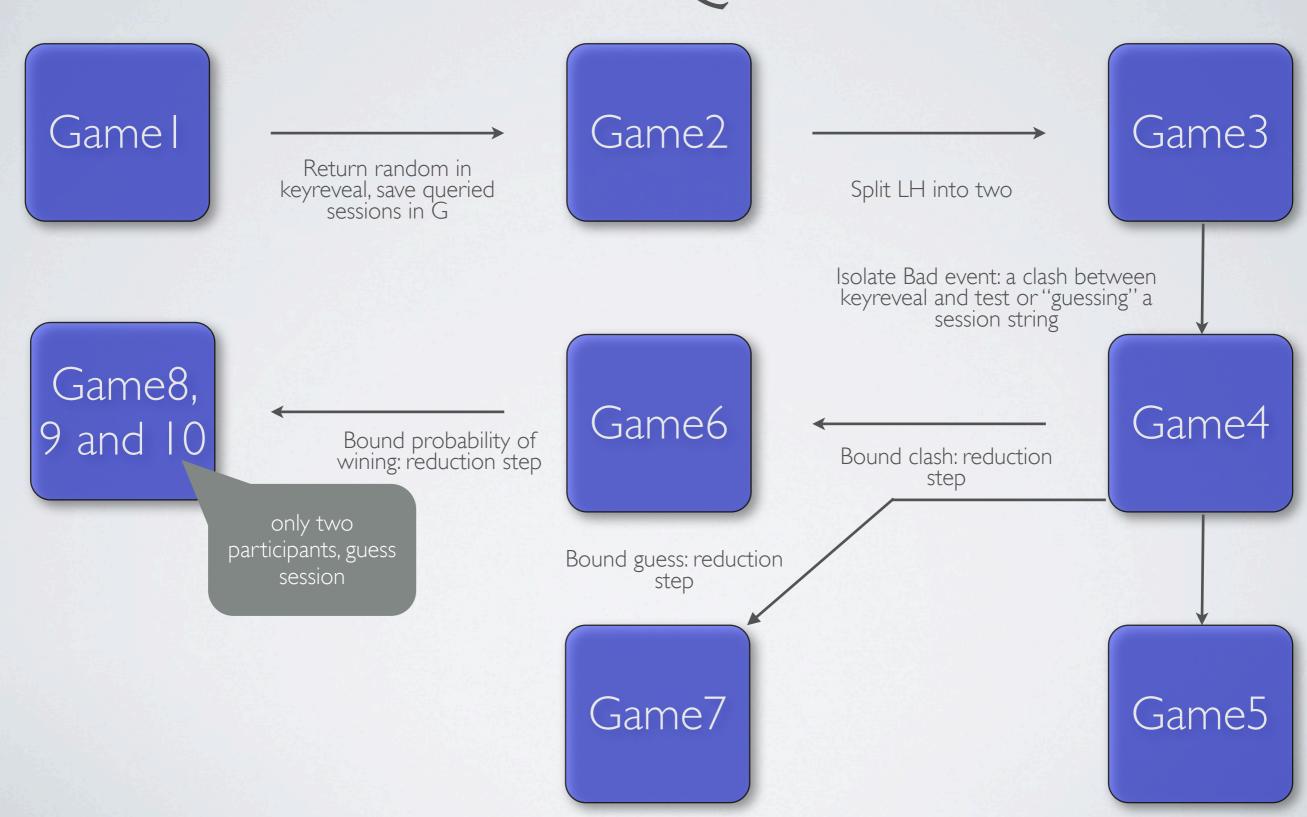
Game3

Game4

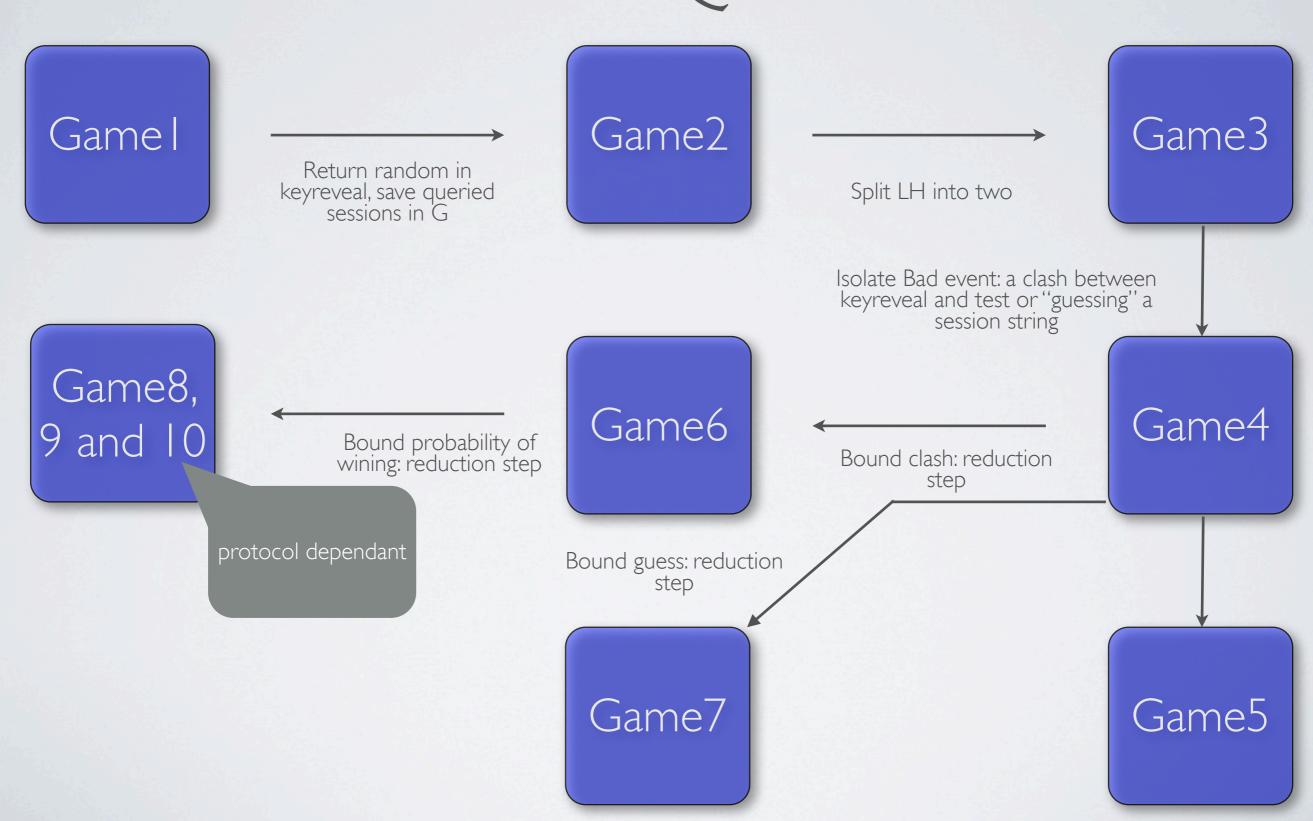
Gamel Game2 Game3 Return random in keyreveal, save queried sessions in G Split LH into two Isolate Bad event: a clash between keyreveal and test or "guessing" a session string Game4 Game6 Bound clash: reduction step Bound guess: reduction step Game7 Game5

Gamel Game2 Game3 Return random in keyreveal, save queried sessions in G Split LH into two Isolate Bad event: a clash between keyreveal and test or "guessing" a session string Game4 Game6 Bound clash: reduction step Bound guess: reduction step Game7 Game5 protocol dependant





Gamel Game3 Game2 Return random in keyreveal, save queried sessions in G Split LH into two Isolate Bad event: a clash between keyreveal and test or "guessing" a session string Game8, Game4 Game6 9 and 10 Bound probability of wining: reduction step Bound clash: reduction step Bound guess: reduction step Game7 Game5



Gamel Game3 Game2 Return random in keyreveal, save queried sessions in G Split LH into two Isolate Bad event: a clash between keyreveal and test or "guessing" a session string Game8, Game4 Game6 9 and 10 Bound probability of wining: reduction step Bound clash: reduction step Bound guess: reduction step Game7 Game5

Gamel Game3 Game2 Return random in keyreveal, save queried sessions in G Split LH into two Isolate Bad event: a clash between keyreveal and test or "guessing" a session string Game8, Game4 Game6 9 and 10 Bound probability of wining: reduction step Bound clash: reduction step Bound guess: reduction step Game7 Game5

# AN INVARIANT (STEP 1-2)

```
invariant1(LH{1},LH{2}) && invariant2(G{2}, LH{1}, skey{2}, seed
{2}) && invariant3(G{2},LH{1}, LH{2}, skey{2}, seed{2}) && ={b,
bad, complete_sessions, incomplete_sessions, corrupt, pkey, skey,
seed, LHT, tested_session,s}
```

```
pred invariant1(...)= forall (str : session_string), in_dom
(str,LH2) => (in_dom(str,LH1) && LH1[str] = LH2[str]
```

```
pred invariant2 (...) = forall (str : session_string, fer_sid :
session_id),(( findelse_g_abs(G2,str,skey2, seed2) <> None) =>
  (in_dom(str,LH1) && LH1[str] = G2[proj(findelse_g_abs
  (G2,str,skey2, seed2))]))
```

```
pred invariant3 (...) = forall (str : session_string), in_dom
(str,LH1) => (in_dom(str, LH2) || findelse_g_abs(G2,str,skey2,seed2) <> None)
```

#### CHALLENGES

- · Scaling: many proof obligations, too much branching.
- Deal with Successive reduction steps in a more succinct way.
- · Identify common proof steps and prove them once and for all.
- Invariant inference.