stateless analysis of a cryptographic protocol

emina torlak · february 22, 2005

authentication

"to be nobody-but-yourself–in a world which is doing its best ... to make you everybody else" - e e cummings

- authentication: verifying the identity of the communicating principals to one another
- authentication protocol: sequence of message exchanges that distributes secrets among principals
- first decentralized authentication protocols described by Needham and Schroeder in 1978:

> *"Finally, protocols such as those developed here are prone to extremely subtle errors that are unlikely to be detected in normal operation. The need for techniques to verify the correctness of such protocols is great, and we encourage those interested in such problems to consider this area."*

needham-schroeder protocol circa 1978

B

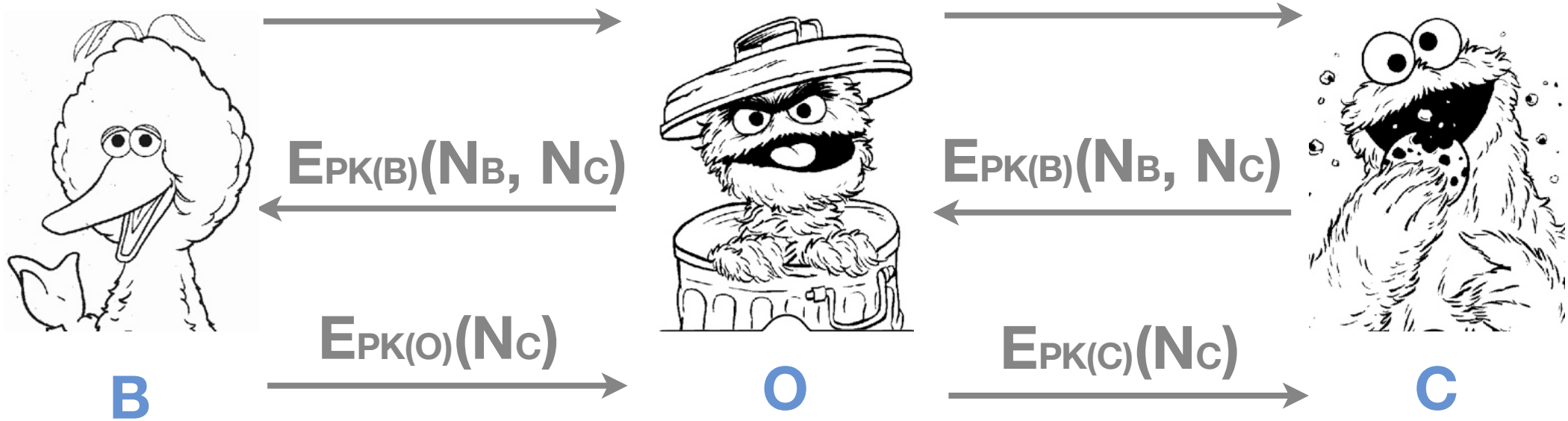
$E_{PK(C)}(I_B, N_B)$

$E_{PK(B)}(N_B, N_C)$

$E_{PK(C)}(N_C)$

C

17 years later... man-in-the-middle attacks



knowledge flow logic

"know or listen to those who know" - baltasar gracian

- preliminaries:
  - P - set of principals
  - V - set of values
  - $k \in K = 2^{P \times V}$ - state of knowledge
  - $R \subseteq P \times V \times P \times K$ - set of communication rules
  - (R, k_0) - knowledge flow
- example: encryption / decryption

$$\forall_{p \in P,\, s,\, v \in V} \left(e, E_{G(s)}(v), p, \{(p, G(s)), (p, v)\}\right)$$

$$\forall_{p \in P,\, s,\, v \in V} \left(e, v, p, \{(p, s), (p, E_{G(s)}(v))\}\right)$$

knowledge flow logic

the importance of being oscar

- project rules on oscar (denoted by o):

  $[X \rightarrow v]$: $\exists_{p \in P-\{o\},\, v \in V} (p, v, o, k) \in R$ where $X = \{v : (o, v) \in k\}$

- example: encryption / decryption

  $\forall_{s,\, v \in V} [\{v, G(s)\} \rightarrow E_{G(s)}(v)]$

  $\forall_{s,\, v \in V} [\{s, E_{G(s)}(v)\} \rightarrow v]$

knowledge flow logic

encoding the needham-schroeder protocol

$$\forall_{p \in P-\{o\},\, p' \in P} \quad [\varnothing \rightarrow E_{G(SK(p'))}(I(p), N(\epsilon, I(p)))]$$

$$\forall_{p \in P-\{o\},\, p' \in P,\, v \in V} \quad [E_{G(SK(p'))}(I(p), v) \rightarrow E_{G(SK(p))}(v, N(E_{G(SK(p'))}(I(p), v), I(p)))]$$

$$\forall_{p \in P-\{o\},\, p' \in P,\, v \in V} \quad [E_{G(SK(p))}(N(\epsilon, I(p)), v) \rightarrow E_{G(SK(p'))}(v)\ ]$$

B

$E_{PK(C)}(I_B, N_B)$

$E_{PK(B)}(N_B, N_C)$

$E_{PK(C)}(N_C)$

C

knowledge flow logic ➠ alloy

principals, values and identities

```
sig Value {}

sig Identity extends Value {}

abstract sig Principal {
   draws :  some Value,
   id       :  some draws & Identity }
{ no id & (Principal - this).@id }

abstract sig HonestUser extends Principal {}
{ draws = Value }

one sig BigBird, CookieMonster extends HonestUser {}

one sig Oscar extends Principal {
   knows :  set Value,
   learns  :  knows->knows }
{ no ^learns & iden }
```

knowledge flow logic ➠ alloy

nonces and ciphertexts

```
sig Nonce extends Value {
   seed :  Value,
   id     : Identity }

sig Ciphertext extends Value {
  plaintext :  some Value,
  key          :  Identity }

pred PerfectCryptography() {
   // each <plaintext, key> pair produces a unique ciphertext
   all disj c1, c2: Ciphertext | c1.plaintext != c2.plaintext || c1.key != c2.key
}
```

knowledge flow logic ➠ alloy

oscar's knowledge

```
pred InitialKnowledge() {
  // Oscar does not draw computed values
  no (Ciphertext + Nonce) & Oscar.draws
}

pred FinalKnowledge() {
  // Oscar knows a value iff he draws it or learns it by communication
  all v:Value | v in (Oscar.draws).*(Oscar.learns) iff v in Oscar.knows
}
```

knowledge flow logic ➠ alloy

primitive rules

```
pred PrimitiveRules(x : set Value, v :Value) {
   // encryption
   (v in Ciphertext && x = v.key + v.plaintext) ||
   // decryption
   (some c : plaintext.v | c.key in Oscar.id && x = (c.key + c)) ||
   // nonce generation
   (v in Nonce && v.id in Oscar.id && x = v.seed)
}
```

knowledge flow logic ➠ alloy

protocol rules

```
pred ProtocolRules(x : set Value, v :Value) {
  v in Ciphertext && {
    // ∅ → EG(SK(p'))(I(p), N(ϵ, I(p)))
    (x :  some Oscar.draws &&
       (let text = v.plaintext, n = text & Nonce |
        #text = 2 && one n && n.seed !in Ciphertext && n.id = text & Identity)) ||

    // EG(SK(p'))(I(p), v) → EG(SK(p))(v, N(r, I(p))) where r = EG(SK(p'))(I(p), v)
    (x : one Ciphertext &&
      (some n : seed.x | #x.plaintext = 2 && v.key in x.plaintext &&
       n.id = x.key && v.plaintext = (x.plaintext - v.key) + n)) ||

    // EG(SK(p))(N(ϵ, I(p)), v) → EG(SK(p'))(v)
    (x : one Ciphertext &&
      (some n : id.(x.key) & Nonce | #x.plaintext = 2 &&
      n in x.plaintext && v.plaintext = x.plaintext - n))
  } }
```

knowledge flow logic ➠ alloy

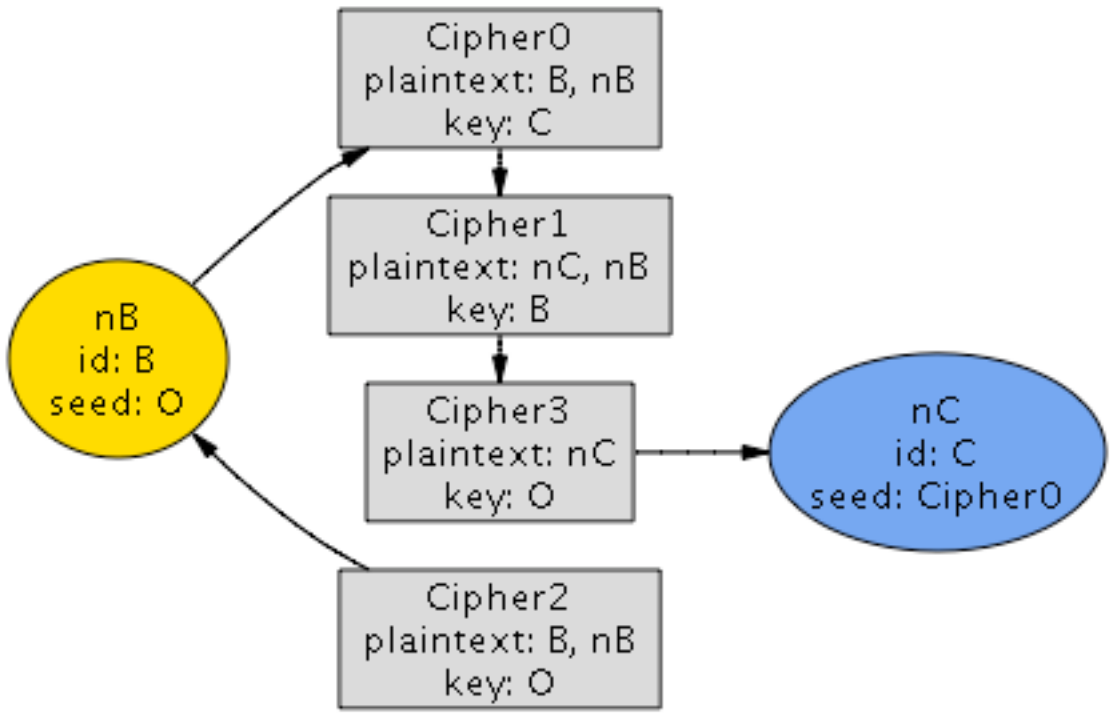
rule application and security theorem

```
pred ApplyRules() {
      all v:Value | let x = Oscar.learns.v |
        some x <=> PrimitiveRules(x,v) || ProtocolRules(x,v)
}

assert NSworks {
  PerfectCryptography() && InitialKnowledge() && FinalKnowledge() && ApplyRules()  =>
  no nB, nC : Oscar.knows & Nonce |
      nB.id in BigBird.id && nC.id in CookieMonster.id &&
      (some c : Ciphertext | nC.seed = c && c.key = nC.id && c.plaintext = nB.id + nB)
}
```

knowledge flow logic ➠ alloy

attack on the needham-schroeder protocol



references

1. Clark, J. and Jacob, J. "A survey of authentication protocol literature". [*manuscript*], Aug 1996.

2. Lowe, G. “An Attack on the Needham-Schroeder Public-Key Authentication Protocol”. *Information Processing Letters*, 56(3), 1995.

3. Needham, R., and Schroeder, M. “Using Encryption for Authentication in Large Networks of Computers”. *Communications of the ACM*, 21(12), Dec 1978.

4. Torlak, E., van Dijk, M., Gassend, B., Kuncak, V., Sachdev, I., Devadas, S. “Knowledge Flow Logic for Modeling and Checking Security Protocols”. [*submitted for publication*], Jan 2005.