02239 Data Security
Security Protocols II

Sebastian Mödersheim

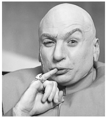
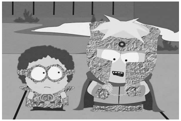
October 2nd, 2024

Outline

Outline

Examples of Intruder Models



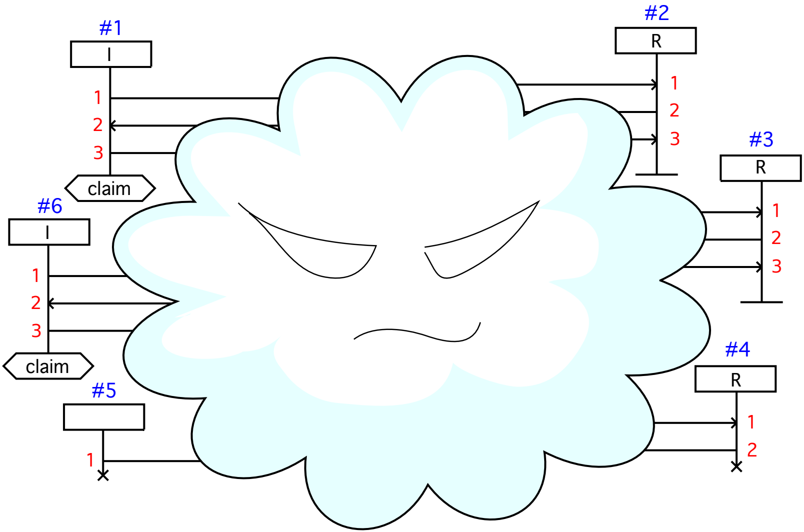



- He knows the protocol but cannot break cryptography.
  (Standard: **perfect encryption**.)
- He is **passive** but overhears all communications.
- He is **active** and can intercept and generate messages.
  "Transfer 100 Kr to Dorrit" $\rightsquigarrow$ "Transfer 10000 Kr to Charlie"
  Worst-case: the intruder controls the entire network
- He might even be one of the principals running the protocol!
  We should expect this unless a role is explicitly assumed to be trusted/honest.

A friend's just an enemy in disguise. You can't trust nobody.

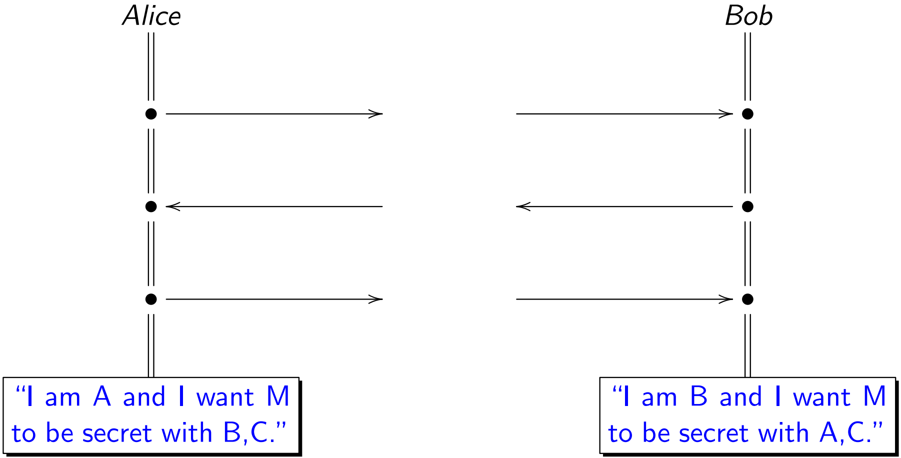
(Charles Dickens, *Oliver Twist*)

Deployment in a Hostile Environment

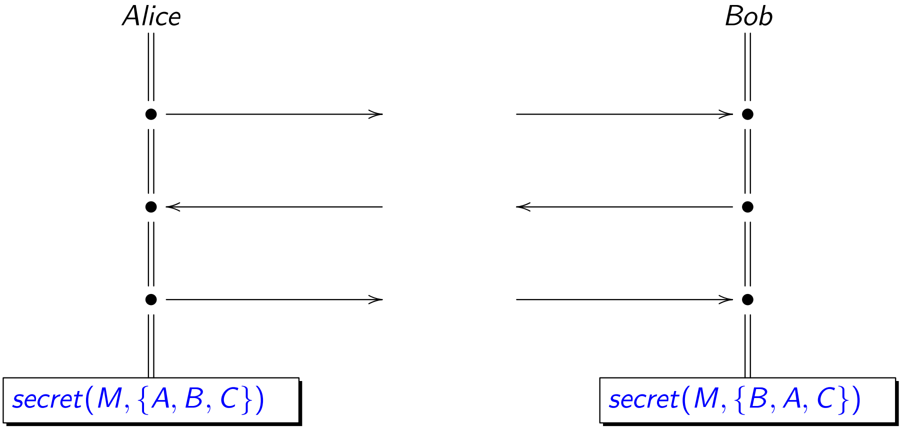


Cloud illustrations from Cas Cremers.

Secrecy: `M secret between A,B,C`

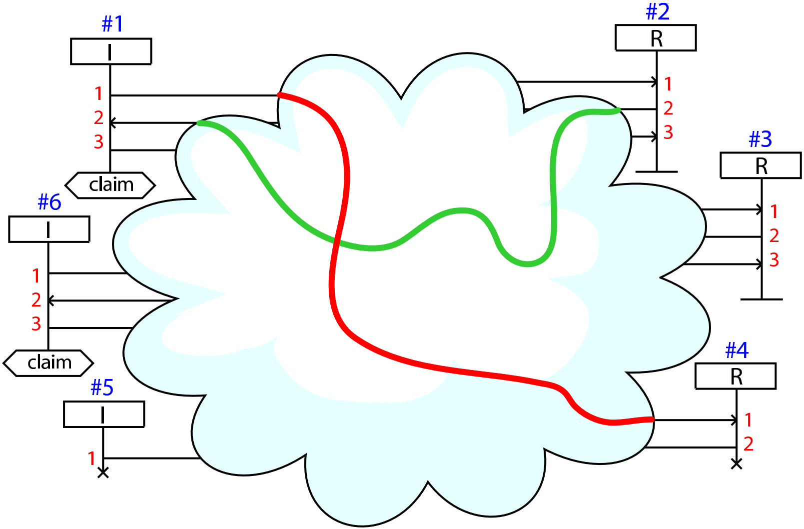


Secrecy: `M secret between A,B,C`

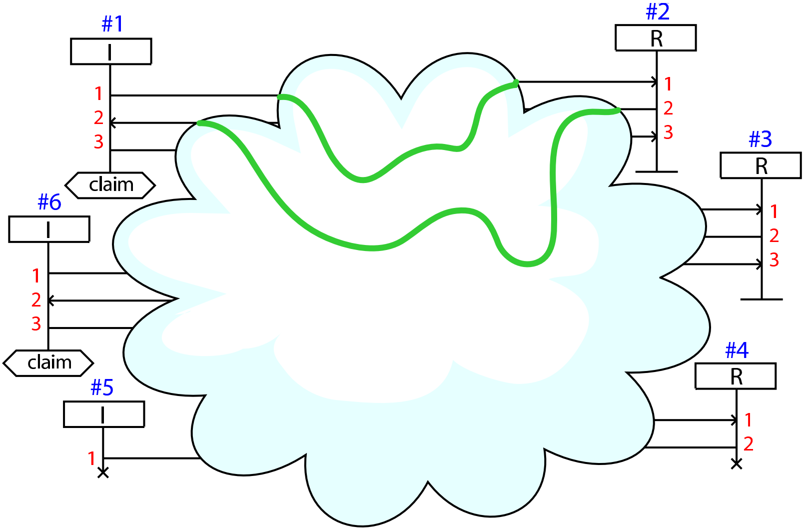


Attack: secret(M,Set), intruder knows M and is not a member of the Set.

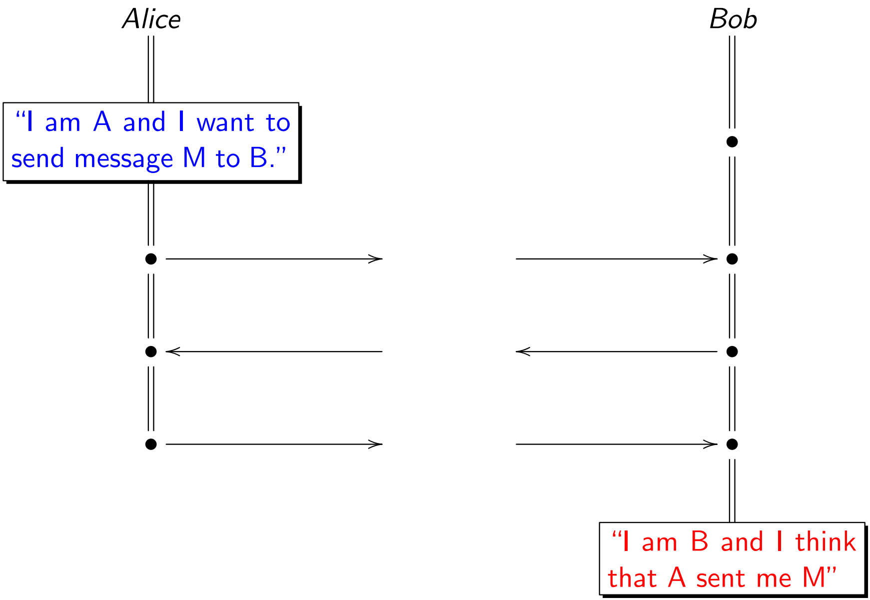
Failed (Weak) Authentication



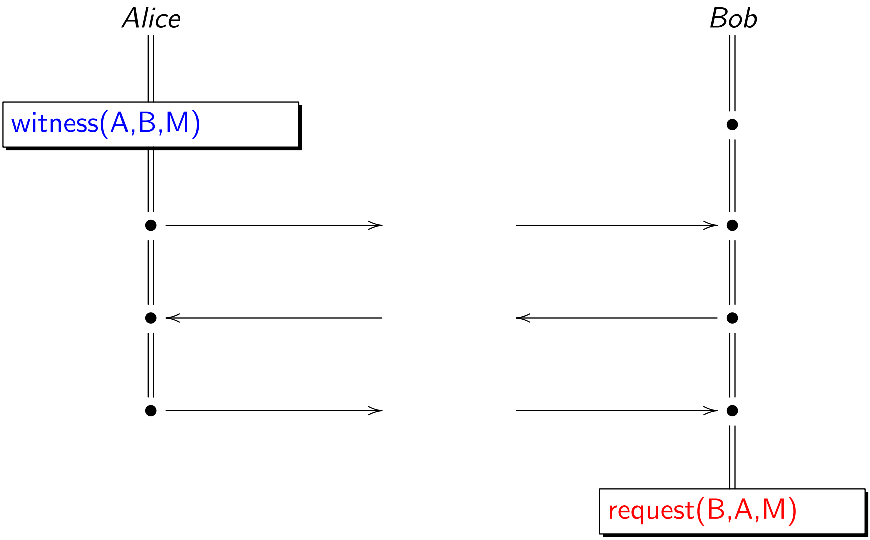
Successful (Weak) Authentication



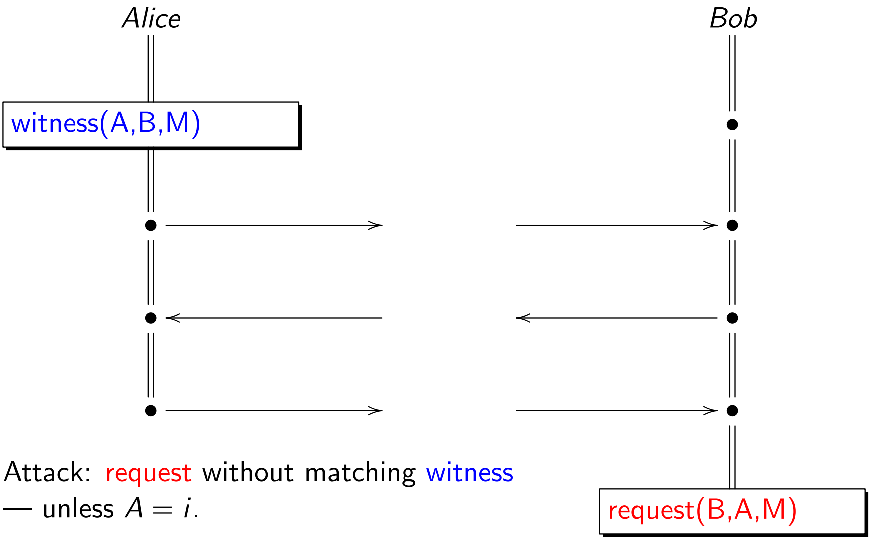
Formalization: (Weak) Authentication



Formalization: (Weak) Authentication

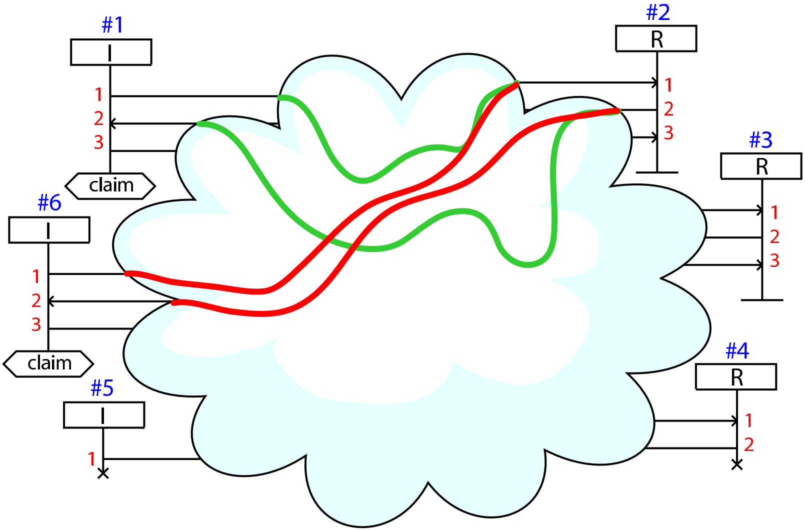


Formalization: (Weak) Authentication

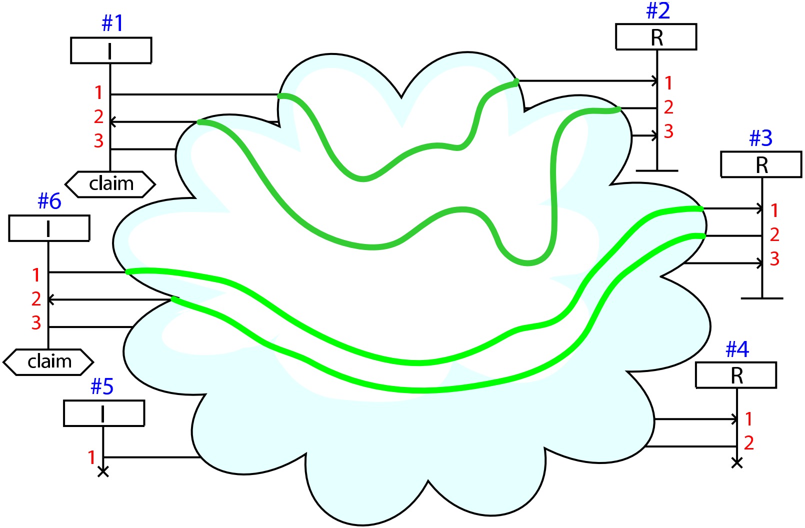


Attack: request without matching witness
— unless $A = i$.

Failed (Strong) Authentication



Successful (Strong) Authentication



Attack: more requests than witnesses.

Outline

Motivation

Example (NSL)

$$\begin{array}{rcll} A & \to & B: & \{NA, A\}_{\mathsf{pk}(B)} \\ B & \to & A: & \{NA, NB, B\}_{\mathsf{pk}(A)} \\ A & \to & B: & \{NB\}_{\mathsf{pk}(B)} \end{array}$$

- Public-key cryptography is used here to ensure confidential transmission of messages.

Motivation

Example (NSL)

$$\begin{array}{llll} A & \to\bullet & B: & NA, A \\ B & \to\bullet & A: & NA, NB, B \\ A & \to\bullet & B: & NB \end{array}$$

- Public-key cryptography is used here to ensure confidential transmission of messages.
- Abstraction: use a confidential channel.

Channel Notation

The Diffie-Hellman assumes an authentic exchange of the half-keys:

$$\begin{array}{cccl} A & \bullet\!\!\rightarrow & B: & \exp(g, X) \\ B & \bullet\!\!\rightarrow & A: & \exp(g, Y) \end{array}$$

- How this exchange is authenticated is not relevant for Diffie-Hellman!
- Many protocols use Diffie-Hellman, e.g. Station2Station, IKE/IKEv2/JFK, Kerberos, TLS, device-pairing. . . .
- Many different ways to authenticate the key-exchange:
  Cryptographically Digital signatures, symmetric/asymmetric encryption, MACs.
  Non-Cryptographically using a trusted third party, meeting face to face, using additional channels (SMS etc.)
- Using an authentic channel abstracts from the realization.

Channel Notation

- Channels can be both assumptions and goals of a protocol:

Example

$$\begin{array}{cccl} A & \bullet\!\!\rightarrow & B & : \ \exp(g, X) \\ B & \bullet\!\!\rightarrow & A & : \ \exp(g, Y) \\ A & \rightarrow & B & : \ \{\!|Payload|\!\}_{\exp(\exp(g,X),Y)} \\ \hline A & \bullet\!\!\rightarrow\!\!\bullet & B & : \ Payload \end{array}$$

"Diffie-Hellman creates a secure channel from authentic channels."

- Actually, public-key cryptography could be defined in a broad sense as a mechanism to obtain secure channels from authentic channels.
- Very general way to see Diffie-Hellman.
- Good for system design and verification: reason about small components with a well-defined interface.

Outline

Transport Layer Security

TLS consists of basically two phases:

- Handshake: A client (typically webbrowser) and a server establish a “secure channel”: a pair of symmetric keys for communicating
- Transport: the parties exchange messages over the secure channel (encrypting with the symmetric keys).

Actually, there are some additions like re-establishing a session which we do not discuss.

TLS 1.3 (simplified)

```
A->B: A,exp(g,X)
B->A: exp(g,Y),
  let k1 = clientK(exp(exp(g,X),Y))
  let k2 = serverK(exp(exp(g,X),Y))
      {| {B,pk(B)}inv(pk(s)) |}k2,
      {| {h(exp(g,X),exp(g,Y))}inv(pk(B)) |}k2
A->B: {|h(exp(g,X),exp(g,Y))|}k1,
      {| data,DATA_A |}k1
B->A: {| data,DATA_B |}k2
```

where

- *h*, *clientK*, *serverK* are one-way functions
- `{B,pk(B)}inv(pk(s))` is a key certificate issued for `B` by a trusted third party `s`.
- `data` is just a tag to distinguish transport messages.
- `DATA_A` and `DATA_B` represent payload messages transmitted on the channel.

TLS 1.3 (simplified)

```
A->B: A,exp(g,X)
B->A: exp(g,Y),
  let k1 = clientK(exp(exp(g,X),Y))
  let k2 = serverK(exp(exp(g,X),Y))
      {| {B,pk(B)}inv(pk(s)) |}k2,
      {| {h(exp(g,X),exp(g,Y))}inv(pk(B)) |}k2
A->B: {|h(exp(g,X),exp(g,Y))|}k1,
      {| data,DATA_A |}k1
B->A: {| data,DATA_B |}k2
```

- Note: only `B` has a certificate, but not `A`.
  - ★ TLS can also be deployed when both sides have a certificate.
  - ★ Typically `A` is a user/webbrowser that does not have a certificate.

TLS 1.3 (simplified)

```
A->B: A,exp(g,X)
B->A: exp(g,Y),
  let k1 = clientK(exp(exp(g,X),Y))
  let k2 = serverK(exp(exp(g,X),Y))
      {| {B,pk(B)}inv(pk(s)) |}k2,
      {| {h(exp(g,X),exp(g,Y))}inv(pk(B)) |}k2
A->B: {|h(exp(g,X),exp(g,Y))|}k1,
      {| data,DATA_A |}k1
B->A: {| data,DATA_B |}k2
```

- Note: only `B` has a certificate, but not `A`.
  - TLS can also be deployed when both sides have a certificate.
  - Typically `A` is a user/webbrowser that does not have a certificate.
- What kind of channel do we get without the client certificate?

TLS 1.3 (simplified)

```
A->B: A,exp(g,X)
B->A: exp(g,Y),
  let k1 = clientK(exp(exp(g,X),Y))
  let k2 = serverK(exp(exp(g,X),Y))
      {| {B,pk(B)}inv(pk(s)) |}k2,
      {| {h(exp(g,X),exp(g,Y))}inv(pk(B)) |}k2
A->B: {|h(exp(g,X),exp(g,Y))|}k1,
      {| data,DATA_A |}k1
B->A: {| data,DATA_B |}k2
```

- Note: only `B` has a certificate, but not `A`.
  - ★ TLS can also be deployed when both sides have a certificate.
  - ★ Typically `A` is a user/webbrowser that does not have a certificate.
- What kind of channel do we get without the client certificate?
- The intruder can impersonate the client A towards B.

TLS 1.3 (simplified)

```
A->B: A,exp(g,X)
B->A: exp(g,Y),
  let k1 = clientK(exp(exp(g,X),Y))
  let k2 = serverK(exp(exp(g,X),Y))
      {| {B,pk(B)}inv(pk(s)) |}k2,
      {| {h(exp(g,X),exp(g,Y))}inv(pk(B)) |}k2
A->B: {|h(exp(g,X),exp(g,Y))|}k1,
      {| data,DATA_A |}k1
B->A: {| data,DATA_B |}k2
```

- Note: only `B` has a certificate, but not `A`.
  - TLS can also be deployed when both sides have a certificate.
  - Typically `A` is a user/webbrowser that does not have a certificate.
- What kind of channel do we get without the client certificate?
- The intruder can impersonate the client A towards B.
- But `B` has a secure channel with whoever created the secret `X`!
  - `A` can be sure who `B` is.
  - `B` cannot be sure who `A` is.

Secure Pseudonymous Channels

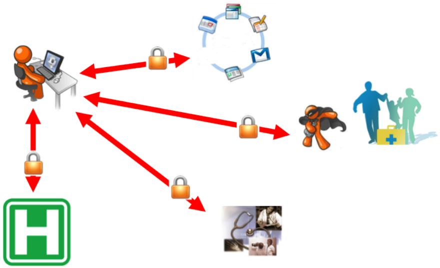
- Consider the $\exp(g, X)$ of agent A as a pseudonym of A.
- The link between A and $\exp(g, X)$ could be achieved by a certificate, but it is not available here.
- The pseudonym $\exp(g, X)$ cannot be stolen/hijacked because "ownership" is the knowledge of x.
- This kind of channel is good enough for many applications such as transmitting credit card data or a login.
- We write often for this kind of channel:

$$[A] \bullet\!\!\rightarrow\!\!\bullet B : M$$

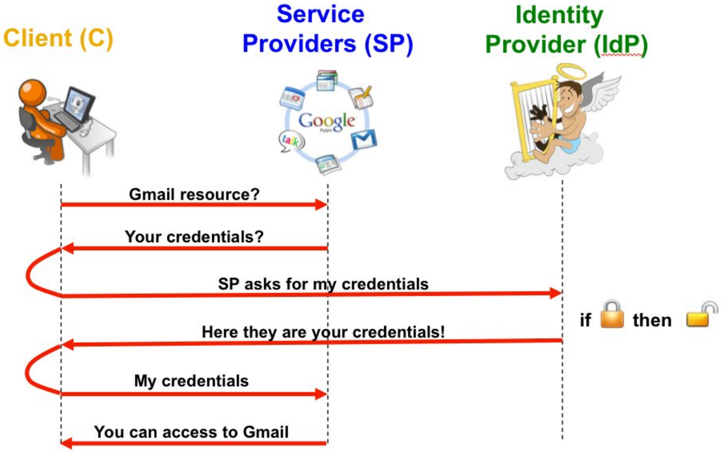
Outline

Single Sign-On (SSO)

- Username/passwords for different websites/accounts:
  - ★ Hassle and security problem
- Avoid this by identity federation:
  - ★ user has trusted identity provider (idp)
  - ★ user signs up only once at idp
  - ★ idp vouches for them to any website (that trusts idp)



SSO with web-browser



Google's SSO

```
Knowledge: C: C,idp,SP,pk(idp);
           idp: C,idp,pk(idp),inv(pk(idp));
           SP: idp,SP,pk(idp)
Actions:
[C] *->* SP  : C,SP,URI
 SP *->* [C] : C,idp,SP,URI,ID

 C  *->* idp : C,idp,SP,URI,ID
idp *->* C   : {C,SP,idp,ID}inv(pk(idp)),URI

[C] *->* SP  : {C,SP,idp,ID}inv(pk(idp)),URI
 SP *->* [C] : Data
Goals:
SP authenticates C on URI
C authenticates SP on Data
Data secret between SP,C
```

Outline

Guessing Attacks
Example: a Variant of Microsoft-ChapV2

```
Protocol: PW
Types: Agent A,B;
       Number NB;
       Function pw,h;
Knowledge:
  A: A,B,pw(A,B),h;
  B: A,B,pw(A,B),h;
Actions:
  B->A: NB
  A->B: h(pw(A,B),NB)
Goals:
B authenticates A on NB
```

What if `pw(A,B)` has low entropy?

Low Entropy Passwords

- Low entropy passwords
  - ★ e.g. natural language words
  - ★ short

Low Entropy Passwords

- Low entropy passwords
  - e.g. natural language words
  - short
- The intruder compiles a dictionary D of passwords
  - Relatively small space of passwords (a few million)
  - Contains many weak passwords

Low Entropy Passwords

- Low entropy passwords
  - e.g. natural language words
  - short
- The intruder compiles a dictionary D of passwords
  - Relatively small space of passwords (a few million)
  - Contains many weak passwords
- Use D for a brute-force attack on an observed message:

Observed message	$nb, h(pw(a,b), nb)$
Construct	$nb, h(\ \ guess\ \ , nb)$ for every $guess \in D$

Low Entropy Passwords

- Low entropy passwords
  - ★ e.g. natural language words
  - ★ short
- The intruder compiles a dictionary D of passwords
  - ★ Relatively small space of passwords (a few million)
  - ★ Contains many weak passwords
- Use D for a brute-force attack on an observed message:

Observed message	$nb, h(pw(a, b), nb)$	
Construct	$nb, h(\ guess\ , nb)$	for every $guess \in D$

- When $pw(a, b) = guess$ for some guess, the constructed and the observed message are identical, and the intruder has found out $pw(a, b)$.

Are weak passwords of any use?

```
Knowledge:
  A: A,B,pw(A,B),pk(B);
  B: A,B,pw(A,B),pk(B),inv(pk(B));
Actions:
  A->B: { A,pw(A,B),K }pk(B)
  B->A: {| NB |}K
Goals:
  B authenticates A on K
  A authenticates B on NB
  K secret between A,B
```

- This is similar to what happens in TLS-based login protocols.

Are weak passwords of any use?

```
Knowledge:
  A: A,B,pw(A,B),pk(B);
  B: A,B,pw(A,B),pk(B),inv(pk(B));
Actions:
  A->B: { A,pw(A,B),K }pk(B)
  B->A: {| NB |}K
Goals:
  B authenticates A on K
  A authenticates B on NB
  K secret between A,B
```

- This is similar to what happens in TLS-based login protocols.
- Guessing not possible – despite low-entropy $pw(A, B)$
  - ★ The intruder has to create $\{A, guess, K\}_{\mathsf{pk}(B)}$ for every guess.
    - ▶ That requires guessing K, too!

Are weak passwords of any use?

```
Knowledge:
  A: A,B,pw(A,B),pk(B);
  B: A,B,pw(A,B),pk(B),inv(pk(B));
Actions:
  A->B: { A,pw(A,B),K }pk(B)
  B->A: {| NB |}K
Goals:
  B authenticates A on K
  A authenticates B on NB
  K secret between A,B
```

- This is similar to what happens in TLS-based login protocols.
- Guessing not possible – despite low-entropy $pw(A, B)$
  - ★ The intruder has to create $\{A, guess, K\}_{\mathsf{pk}(B)}$ for every guess.
    - ▶ That requires guessing K, too!
- This is also why any reasonable implementation of asymmetric encryption contains randomization!
  - ★ $\{guessable\ message\}_{\mathsf{pk}(B)}$

Guessing Attacks in OFMC

```
B->A: NB
A->B: h(pw(A,B),NB)
Goals:
B authenticates A on NB
pw(A,B) guessable secret between A,B
```

Gives attack:

```
GOAL:
  guesswhat
...
ATTACK TRACE:
i -> (x20,1): x205
(x20,1) -> i: h(pw(x20,x25),x205)
i can produce secret h(guessPW,x206)
```

Guessing Attacks in OFMC

Implementation in OFMC:

- Say $pw(A, B)$ is specified as a guessable secret between A and B.
- Let $guessPW$ be a constant known to the intruder.
- Check every message sent by an honest agent if it contains $pw(A, B)$.
  - ★ Example: outgoing message $h(pw(A, B), NA)$
  - ★ Then declare $h(guessPW, NA)$ as a secret between A and B.
  - ★ If the intruder is able to violate this secrecy goal, then he can make a guessing attack.
- Additionally, if the password is used for symmetric encryption, then guessing the key is sufficient:
  - ★ Example: $\{|m|\}_{h(pw(A,B),NA)}$
  - ★ Then also $h(guessPW, NA)$ is a secret between A and B.