# The security protocols library digest

## Andrew Secure RPC

**Author(s):** M. Satyanarayanan 1987
*Last modified November 14, 2011*

**Summary:** Exchanged of a fresh shared key. Symmetric key cryptography.

### Protocol specification (in common syntax)

```
A, B :          principal
Kab, K'ab :     symkey
Na, Nb, N'b :   nonce
succ :          nonce -> nonce

1.    A   ->   B   :    A, {Na}Kab
2.    B   ->   A   :    {succNa, Nb}Kab
3.    A   ->   B   :    {succNb}Kab
4.    B   ->   A   :    {K'ab, N'b}Kab
```

### Description of the protocol rules

This protocol establishes the fresh shared symmetric key `K'ab`. The nonce `N'b` is sent in message 4 to be used in a future session.

We assume that initially, the symmetric keys `Kab` is known only to `A` and `B`.

### Requirements

The protocol must guaranty the secrecy of the new shared key `K'ab`: in every session, the value of `K'ab` must be known only by the participants playing the roles of `A` and `B`.

The protocol must guaranty the authenticity of `K'ab`: in every session, on reception of message 4, `A` must be ensured that the key `K'ab` in the message has been created by `A` in the same session.

### References

[Sat89]

### Claimed attacks

[BAN89]. The message 4 contains nothing that A knows to be fresh. Hence, an intruder I can replay this message in another session of the protocol to convinced B to accept an old compromised key.

```
i.1.     A     ->    B     :   A, {Na}Kab
i.2.     B     ->    A     :   {succNa, Nb}Kab
i.3.     A     ->    B     :   {succNb}Kab
i.4.     B     ->    A     :   {K'ab, N'b}Kab
ii.1.    A     ->    B     :   A, {Ma}Kab
ii.2.    B     ->    A     :   {succMa, Mb}Kab
ii.3.    A     ->    B     :   {succMb}Kab
ii.4.    B     ->   I(A)   :   {K''ab, M'b}Kab
ii.4.   I(B)   ->    A     :   {K'ab, N'b}Kab
```

### See also

BAN modified Andrew Secure RPC, BAN concrete Andrew Secure RPC, Lowe modified BAN concrete Andrew Secure RPC.

## BAN modified Andrew Secure RPC

**Author(s):** Michael Burrows and Martin Abadi and Roger Needham 1987
*Last modified November 14, 2002*

**Summary:** Modified version of Andrew Secure RPC correcting a freshness flaw. Exchanged of a fresh shared key, Symmetric key cryptography.

### Protocol specification (in common syntax)

```
A, B :          principal
Kab, K'ab :     symkey
Na, Nb, N'b :   nonce
succ :          nonce -> nonce

1.   A  ->  B  :   A, {Na}Kab
2.   B  ->  A  :   {succNa, Nb}Kab
3.   A  ->  B  :   {succNb}Kab
4.   B  ->  A  :   {K'ab, N'b, Na}Kab
```

### Description of the protocol rules

The nonce `Na` has been added to the message `4` of Andrew Secure RPC to prevent the flow presented in Andrew Secure RPC.

### Requirements

See Andrew Secure RPC.

### References

[BAN89]

### See also

Andrew Secure RPC, BAN concrete Andrew Secure RPC, Lowe modified BAN concrete Andrew Secure RPC.

## BAN concrete Andrew Secure RPC

**Author(s):** Michael Burrows and Martin Abadi and Roger Needham 1989
*Last modified November 14, 2002*

**Summary:** A concrete realization of the Andrew Secure RPC protocol, stronger and with less encryption. Exchanged of a fresh shared key, Symmetric key cryptography.

### Protocol specification (in common syntax)

```
A, B :          principal
Kab, K'ab :     symkey
Na, Nb, N'b :   nonce
succ :          nonce -> nonce

1.    A  ->  B   :    A, Na
2.    B  ->  A   :    {Na, K'ab}Kab
3.    A  ->  B   :    {Na}K'ab
4.    B  ->  A   :    Nb
```

### Description of the protocol rules

This protocol establishes the fresh shared symmetric key `K'ab`.

The nonce `Nb` is sent in message `4` to be used in a future session.

We assume that initially, the symmetric keys `Kab` is known only to `A` and `B`.

### Requirements

See Andrew Secure RPC.

### References

[BAN89]

### Claimed attacks

In [Low96], with 2 parallel runs where the intruder `I` impersonates `B`.

```
 i.1.      A     ->  I(B)   :    A, Na
 ii.1.    I(B)   ->   A     :    B, Na
 ii.2.     A     ->  I(B)   :    {Na, K'ab}Kab
 i.2.     I(B)   ->   A     :    {Na, K'ab}Kab
 i.3.      A     ->  I(B)   :    {Na}K'ab
 ii.3.    I(B)   ->   A     :    {Na}K'ab
 i.4.     I(B)   ->   A     :    Ni
 ii.4.     A     ->  I(B)   :    Nb
```

A fix to this attack can be found in Lowe modified BAN concrete Andrew Secure RPC.

### See also

Andrew Secure RPC, BAN modified Andrew Secure RPC, Lowe modified BAN concrete Andrew Secure RPC.

## Lowe modified BAN concrete Andrew Secure RPC

**Author(s):** Gavin Lowe 1996
*Last modified November 14, 2002*

**Summary:** A modified version of the BAN concrete Andrew Secure RPC protocol, preventing a parallel session attack. Exchanged of a fresh shared key, Symmetric key cryptography.

### Protocol specification (in common syntax)

```
A, B :          principal
Kab, K'ab :     symkey
Na, Nb, N'b :   nonce
succ :          nonce -> nonce

1.    A  ->  B   :    A, Na
2.    B  ->  A   :    {Na, K'ab, B}Kab
3.    A  ->  B   :    {Na}K'ab
4.    B  ->  A   :    Nb
```

### Description of the protocol rules

The identity of the responder `B` has been added in the message `2` of `andrewBAN2`.

### Requirements

See Andrew Secure RPC.

### References

[Low96]

### See also

Andrew Secure RPC, BAN modified Andrew Secure RPC, BAN concrete Andrew Secure RPC.

## Bull's Authentication Protocol

**Author(s):** J. Bull 1997

**Summary:** This protocol, described in [BO97], aims at establishing fresh session keys between a fixed number of participants (for instance 3) and a server: one key for each pair of agents adjacent in the chain.

## Protocol specification (in common syntax)

```
A, B, C, S :     principal
Kab, Kbc :       fresh symkey
Na, Nb, Nc :     fresh number
Kas, Kbs, Kcs :  symkey
h :              message, symkey -> message
```

```
A computes Xa = h((A,B,Na),Kas), (A,B,Na)

 1.    A  ->  B   :    Xa

B computes Xb = h((B,C,Nb,Xa),Kbs), (B,C,Nb,Xa)

 2.    B  ->  C   :    Xb

C computes Xc = h((C,S,Nc,Xb),Kcs), (C,S,Nc,Xb)

 3.    C  ->  S   :    Xc

 4.    S  ->  C   :    A, B, Kab xor h(Na,Kas), {A,B,Na}Kab,
                       B, A, Kab xor h(Nb,Kbs), {B,A,Nb}Kab,
                       B, C, Kbc xor h(Nb,Kbs), {B,C,Nb}Kbc,
                       C, B, Kbc xor h(Nc,Kcs), {C,B,Nc}Kbc
 5.    C  ->  B   :    A, B, Kab xor h(Na,Kas), {A,B,Na}Kab,
                       B, A, Kab xor h(Nb,Kbs), {B,A,Nb}Kab,
                       B, C, Kbc xor h(Nb,Kbs), {B,C,Nb}Kbc
 6.    B  ->  A   :    A, B, Kab xor h(Na,Kas), {A,B,Na}Kab
```

## Description of the protocol rules

The protocol is initiated by A and then goes through B and C before reaching S. At the end, new session keys Kab and Kbc are established. The properties of exclusive or are:

$$\begin{aligned} \texttt{x xor (y xor z)} &= \texttt{(x xor y) xor z} \quad \text{(E1)} \\ \texttt{x xor y} &= \texttt{y xor x} \quad \text{(E2)} \\ \texttt{x xor 0} &= \texttt{x} \quad \text{(E3)} \\ \texttt{x xor x} &= \texttt{0} \quad \text{(E4)} \end{aligned}$$

## Requirements

The protocol must guaranty the secrecy of Kxy. Each key Kxy should be known to exactly x and y (and also S), even if some nodes other than x and y are malicious.

### References

[BO97]

### Claimed attacks

This protocol is subject to an attack [RS98] that can be mounted by a dishonest participant. For example, assume that C is a malicious agent. He can intercept Kab xor h(Nb,Kbs) and Kbc xor h(Nb,Kbs) sent by S at step 4, and since C knows the session key Kbc, he can compute Kbc xor Kab xor h(Nb,Kbs) xor Kbc xor h(Nb,Kbs). Since this term is actually equal to Kab, the agent C learns a session key that should be shared only by A and B.

## CAM

**Author(s):** Greg O'Shea and Michael Roe April 2001
*Submitted by Michael Roe January 10, 2003*
*Last modified November 28, 2002*

**Summary:** A protocol used by mobile computers to inform their peers when their network address has changed.

### Protocol specification (in common syntax)

```
M,C :    principal
Tm :     timestamp
PK,SK :  principal -> key (keypair)
HoA :    principal -> address
CoA :    principal -> address
i :      salt

1.   M  ->  C  :   CoA(M), HoA(C), HoA(M), PK(M), i, Tm,
                   {H(CoA(M), HoA(C), HoA(M), Tm)}SK(M)
     HostPart(HoA(M)) = H(PK(M), i)
```

### Description of the protocol rules

Each mobile node (M) generates a key pair PK(M), SK(M). M then generates a home address HoA(M) by concatenating the routing prefix of its home network with a hash of PK(M) and a salt i. HoA(M) serves two purposes.

It is used by the correspondent `C` as an identifier for `M`, and it is a routable network address that can be used to contact a home agent that will forward messages on to `M`. The places where `M` can be attached to the network are also given identifiers; `CoA(M)` is the identifier of `M`'s current network attachment point. `CoA(M)` varies over time. `M` knows (by means outside the protocol) when `CoA(M)` changes.

`M` has a set of correspondents that it wishes to communicate with. The set of `M`'s correspondents varies over time.

`M` runs the protocol with `C` when any of these events happens:

- `CoA(M)` changes and `C` is one of `M`'s correspondents
- `M` adds `C` to its set of correspondents
- `C` is one of `M`'s correspondents, and time `delta1T` (as measured by `M`'s local clock) has elapsed since `M` last ran the protocol with `C`

Each correspondent `C` maintains a table mapping home addresses `HoA(M)` to care-of addresses `CoA(M)`. This is a partial table — there can be home addresses `HoA(M)` that do not have an entry in the table.

When `C` receives message `1`, it will check that the timestamp `Tm` is within `delta2T` of the current time (as measured by `C`'s local clock); that the home address satisfies the relation `HostPart(HoA(M)) = H(PK(M), i)`; and that the signature can be verified with `PK(M)`. If all of these checks pass, `C` adds the pair to `(HoA(M),CoA(M))` to its table, replacing the previous entry for `HoA(M)` if one exists.

If `C` has not accepted a valid message containing `HoA(M)` within the last `Delta3T` seconds, then it will remove the entry for `HoA(M)` from its table.

The local clocks of `M` and `C` are assumed to be loosely synchronised. That is, there exists a `Delta4T` such that the times measured by `C` and `M`'s clocks are within `Delta4T` of each other. Clocks are assumed to be monotonically increasing.

## Requirements

There is a time interval `DeltaT` such that if `CoA(M)` has not changed within the last `DeltaT` seconds, and both `C` and `M` are following the protocol, then either `C`'s table does not contain an entry for `HoA(M)` or `C`'s table contains `(HoA(M), CoA(M))`.

### References

This protocol was described by O'Shea and Roe in Computer Communications Review [OR01]. A concrete realisation of this protocol is given in the first version of the Internet draft `draft-roe-mobileip-updateauth-00.txt` ([RAOA02]); later versions of this document describe a different protocol that meets additional requirements. The idea of constructing IPv6 addresses from the hash of a public key was proposed by Christian Huitema [Hui98], Jeff Schiller and others.

Related protocols have been proposed by Bradner, Mankin and Schiller [BMS02], Montenegro and Castelluccia [MC02] and Nikander [Nik01, NYW03].

### Remark

Authentication of the principal `M` is not a goal of this protocol. Although `C` cannot necessarily distinguish a run of the protocol with `M` from a run of the protocol with a different principal, this is not an attack.

If authentication of `M` is desired, the protocol can be used in conjunction with an additional protocol that authenticates `M`.

Runs of the protocol in which `M` tries to run the protocol with `C`, but `C` does not create a table entry (e.g. because an attacker prevents the message from reaching `C`) are also not attacks. It is an assumption of the protocol that the absence of a table entry for `HoA(M)` is "fail safe" and does not correspond to an insecure state. The table entry is used for an optimisation only; if it is not present, `C` has an alternative method of proceeding without it.

## CCITT X.509 (1)

**Author(s):** CCITT 1987
*Last modified November 22, 2002*

**Summary:** One message protocol from the recommendations of the CCITT for the CCITT.X.509 standard.

### Remark

This protocol presented here is actually a simplified version from [BAN89] and [AN96].

## Protocol specification (in common syntax)

```
A, B :    principal
Na, Nb :  nonce
Ta, Tb :  timestamp
Ya, Yb :  userdata
Xa, Xb :  userdata
PK, SK :  principal -> key (keypair)

1.    A  ->  B   :    A, {Ta, Na, B, Xa, {Ya}PK(B)}SK(A)
```

## Description of the protocol rules

The timestamp `Ta` and nonce `Na` are not used here.

`Xa` and `Ya` are the data transmitted, the privacy of `Ya` is ensured by its encryption with the public key of `B` and the authenticity of `Xa` and `Ya` is ensured by the encryption with the private key of `A`.

## Remark

As explained in [BAN89], in the original protocol specification [CCI87], only a hash of the data is signed, for efficiency reasons. This means that the message should be specified by:

```
1.    A  ->  B   :    A, Ta, Na, B, Xa, {Ya}PK(B), {h(Ta, Na, B, Xa, {Ya}PK(B))}SK(A
```

where `h` is a one-way function.

## Requirements

The protocol must ensure the confidentiality of `Ya`: if `A` and `B` follow the protocol, then an attacker should not be able to obtain `Ya`.

The protocol must ensure the recipient `B` of the message that the data `Xa` and `Ya` originate from `A`.

## References

[CCI87], [BAN89].

## Claimed attacks

[AN96]. Failure of the authenticity of `Xa` and `Ya`.

```
i.1.     A  ->  I(B)   :   A, {Ta, Na, B, Xa, {Ya}PK(B)}SK(A)
ii.1.    I  ->   B     :   I, {Ta, Na, B, Xa, {Ya}PK(B)}SK(I)
```

### See also

CCITT X.509 (1c), CCITT X.509 (3).

### Comment sent by Michael Roe (*20/11/2002*)

The requirements section should include a confidentiality property: if A and B follow the protocol, then an attacker should not be able to obtain Ya.

*Note of the moderator: this property has been added above, following the comment.*

### Comment sent by Michael Roe (*20/11/2002*)

In this protocol, private keys are used for different two operations: digital signature and message decryption. The processing done with the key is not the same in the two cases, and the difference can matter in protocol verification. The notation ought to distinguish the two operations

## CCITT X.509 (1c)

**Author(s):** M. Abadi and R Needham 1996
*Last modified November 11, 2002*

**Summary:** Correction of the CCITT X.509 (1) one message protocol.

### Protocol specification (in common syntax)

```
A, B :    principal
Na, Nb :  nonce
Ta, Tb :  timestamp
Ya, Yb :  userdata
Xa, Xb :  userdata
PK, SK :  principal -> key (keypair)
h :       userdata -> userdata (one-way)

1.   A  ->  B  :   A, {Ta, Na, B, Xa, {Ya, {h(Ya)}SK(A)}PK(B)}SK(A)
```

### Description of the protocol rules

See CCITT X.509 (1). The solution proposed in [AN96] to correct the authentication flaw in the CCITT X.509 (1) one message protocol is to sign the secret data `Ya` before it is encrypted.

### Requirements

The protocol must ensure the recipient `B` of the message that the data `Xa` and `Ya` originate from `A`.

### References

[AN96], [CCI87].

### See also

CCITT X.509 (1), CCITT X.509 (3).

## CCITT X.509 (3)

**Author(s):** CCITT 1987
*Last modified November 22, 2002*

**Summary:** Three messages protocol in the recommendations of the CCITT for the CCITT.X.509 standard.

### Remark

This protocol presented here is actually a simplified version from [BAN89] and [AN96].

### Protocol specification (in common syntax)

```
A, B :    principal
Na, Nb :  nonce
Ta, Tb :  timestamp
Ya, Yb :  userdata
Xa, Xb :  userdata
PK, SK :  principal -> key (keypair)
```

```
 1.    A  ->  B  :    A, {Ta, Na, B, Xa, {Ya}PK(B)}SK(A)
 2.    B  ->  A  :    B, {Tb, Nb, A, Na, Xb, {Yb}PK(A)}SK(B)
 3.    A  ->  B  :    A, {Nb}SK(A)
```

### Description of the protocol rules

See CCITT X.509 (1).

### Remark

As in the case of CCITT X.509 (1), in the original protocol specification [CCI87], only a hash of the data is signed, for efficiency reasons. Hence the messages specification ought to be:

```
 1.    A  ->  B  :    A, Ta, Na, B, Xa, {Ya}PK(B), {h(Ta, Na, B, Xa, {Ya}PK(B))}SK(A
 2.    B  ->  A  :    B, Tb, Nb, A, Na, Xb, {Yb}PK(A), {h(B, Tb, Nb, A, Na, Xb, {Yb}
 3.    A  ->  B  :    A, {Nb}SK(A)
```

where h is a one-way function.

### Requirements

The protocol must ensure the confidentiality of Ya and Yb: if A and B follow the protocol, then an attacker should not be able to obtain Ya or Yb.

The protocol must ensure the recipient B of the message 1 that the data Xa and Ya originate from A.

The protocol must ensure the recipient A of the message 2 that the data Xb and Yb originate from B.

### References

[BAN89], [CCI87].

### Claimed attacks

**1.** This parallel session attack presented in [BAN89] works if B does not check the timestamp Ta in the first message.

```
i.1.      A     ->  I(B)   :   A, {Ta, Na, B, Xa, {Ya}PK(B)}SK(A)
i.1.     I(A)   ->   B     :   A, {Ta, Na, B, Xa, {Ya}PK(B)}SK(A)
i.2.      B     ->  I(A)   :   B, {Tb, Nb, A, Na, Xb, {Yb}PK(A)}SK(B)
ii.1.     A     ->   I     :   A, {Ta', Na', C, Xa', {Ya'}PK(I)}SK(A)
ii.2.     I     ->   A     :   I, {Ti, Nb, A, N'a,Xi, {Yi}PK(A)}SK(I)
ii.3.     A     ->   I     :   A, {Nb}SK(A)
ii.3.    I(A)   ->   B     :   A, {Nb}SK(A)
```

**2.** Another attack can be found in [lM90].

### See also

CCITT X.509 (1), CCITT X.509 (1c), BAN modified version of CCITT X.509 (3).

## BAN modified version of CCITT X.509 (3)

**Author(s):** Michael Burrows and Martin Abadi and Roger Needham 1989
*Last modified January 16, 2003*

**Summary:** Modified version of the three messages protocol in the recommendations of the CCITT for the CCITT.X.509 standard (CCITT X.509 (3)).

### Protocol specification (in common syntax)

```
A, B :    principal
Na, Nb :  nonce
Ya, Yb :  userdata
Xa, Xb :  userdata
PK, SK :  principal -> key (keypair)

1.    A  ->  B   :   A, {Na, B, Xa, {Ya}PK(B)}SK(A)
2.    B  ->  A   :   B, {Nb, A, Na, Xb, {Yb}PK(A)}SK(B)
3.    A  ->  B   :   A, {B, Nb}SK(A)
```

### Description of the protocol rules

Compared to CCITT X.509 (3), the identity of `B` has been added to the signature in message `3`. This prevents the [BAN89] attack on the CCITT X.509

(3) protocol, which can occur when B does not check the timestamps. With this modification, the timestamps become redundant and can be removed.

### Requirements

See CCITT X.509 (3).

### References

[BAN89].

### See also

CCITT X.509 (1), CCITT X.509 (1c), CCITT X.509 (3).

## Denning-Sacco shared key

**Author(s):** Dorothy E. Denning and Giovanni Maria Sacco 1981
*Last modified November 12, 2002*

**Summary:** Modified version of the Needham Schroeder Symmetric Key with timestamps to fix the freshness flaw. Distribution of a shared symmetric key by a trusted server and mutual authentification. Symmetric key cryptography with server and timestamps.

### Protocol specification (in common syntax)

```
A, B, S :         principal
Kas, Kbs, Kab :   key
T :               timestamp

1.   A  ->  S  :   A, B
2.   S  ->  A  :   {B, Kab, T, {Kab, A, T}Kbs}Kas
3.   A  ->  B  :   {Kab,A, T}Kbs
```

### Description of the protocol rules

The nonces of Needham Schroeder Symmetric Key (for mutual authentication of A and B) have been replaced by a timestamp T.

The shared symmetric key established by the protocol is Kab.

### Requirements

See Needham Schroeder Symmetric Key.

### References

[DS81]

### Claimed attacks

This protocol is subject to a mutiplicity attack [Low97].

```
i.1.      A    ->  S  :  A, B
i.2.      S    ->  A  :  {B, Kab, T, {Kab, A, T}Kbs}Kas
i.3.      A    ->  B  :  {Kab,A, T}Kbs
ii.3.   I(A)   ->  B  :  {Kab,A, T}Kbs
```

In session ii, B thinks that A wants to establish a new shared key and accepts it.

### See also

Lowe modified Denning-Sacco shared key, Needham Schroeder Symmetric Key.

## Lowe modified Denning-Sacco shared key

**Author(s):** Gavin Lowe 1997
*Last modified November 12, 2002*

**Summary:** Modified version of the Denning-Sacco shared key protocol to correct a freshness flaw. Distribution of a shared symmetric key by a trusted server and mutual authentification. Symmetric key cryptography with server and timestamps.

### Protocol specification (in common syntax)

```
A, B, S :         principal
Nb :              nonce
Kas, Kbs, Kab :   key
T :               timestamp
dec :             nonce -> nonce
```

```
1.   A  ->  S  :   A, B
2.   S  ->  A  :   {B, Kab, T, {Kab, A, T}Kbs}Kas
3.   A  ->  B  :   {Kab,A, T}Kbs
4.   B  ->  A  :   {Nb}Kab
5.   A  ->  B  :   {dec(Nb)}Kab
```

### Description of the protocol rules

This version add a nonce handshake (messages 4, 5) at the end of Denning-Sacco shared key to prevent the attack from [Low97].

### Requirements

See Needham Schroeder Symmetric Key.

### References

[Low97]

### See also

Needham Schroeder Symmetric Key, Denning-Sacco shared key.

## Diffie Helman

**Author(s):** W. Diffie and M. Helman 1978
*Last modified November 11, 2002*

**Summary:** The Diffie Helman key exchange algorithm.

### Protocol specification (in common syntax)

```
A, B :          principal
P, G, Xa, Xb :  number
one :            -> number
kap :           number, number, number -> number

1.   A  ->  B  :   P, G
2.   A  ->  B  :   kap(P, G, Xa)
3.   B  ->  A  :   kap(P, G, Xb)
4.   A  ->  B  :   {one()}kap(P, kap(P, G, Xb), Xa)
```

### Description of the protocol rules

The function `kap` must satisfy:

`kap(P, kap(P, G, Y), X)` = `kap(P, kap(P, G, X), Y)`

It is implemented by: `kap(P, X, Y) = exp(X, Y) mod P`.

It the protocol, `P` is choosen to be a prime number `P` and `G < P`.

The fresh symmetric key exchanged is `kap(P,kap(P,G,Xb),Xa)` = `kap(P,kap(P,G,Xa),Xb)`.

### Requirements

The protocol must guaranty the secrecy of the fresh key.

The protocol must guaranty the authenticity of the participants.

### References

[DH76]

### Claimed proofs

[Bla01]

### Claimed attacks

The authenticity is not guaranteed by the protocol.

```
1.  I(A)  ->   B    :  P, G
2.  I(A)  ->   B    :  kap(P, G, Xi)
3.   B    ->  I(A)  :  kap(P, G, Xb)
4.  I(A)  ->   B    :  {one()}kap(P, kap(P, G, Xb), Xi)
```

or

```
1.   A    ->  I(B)  :  P, G
2.   A    ->  I(B)  :  kap(P, G, Xa)
3.  I(B)  ->   A    :  kap(P, G, Xi)
4.   A    ->  I(B)  :  {one()}kap(P, kap(P, G, Xi), Xa)
```

## GJM

**Author(s):** Juan A. Garay, Markus Jakobson, Philip MacKenzie 1999

*Submitted by Alexandre Boisseau October 30, 2002*

**Summary:** The goal of this protocol is to achieve distributed contract signing in an abuse-free way, that is no party ever can prove to a third party that he is able of determining the issue of the exchange (validate or invalidate the contract). To achieve this goal, a special construction called private contract signature is introduced. Such a private contract signature has the particular property that it is meaningful only for a given trusted third party. Moreover, this protocol is optimistic in the sense that the trusted third party is required only in case of problem.

## Protocol specification (in common syntax)

```
A,B,T :            principal
C :                msg
PCS :              (principal,msg,principal,principal):msg
S-SIG :            (principal,msg):msg
TP-SIG :           (principal,msg):msg
resolved,aborted : bool
abort :            msg

Exchange-1.    A  ->  B   :   PCS(A,C,B,T)
Exchange-2.    B  ->  A   :   PCS(B,C,A,T)
Exchange-3.    A  ->  B   :   S-SIG(A,C)
Exchange-4.    B  ->  A   :   S-SIG(B,C)
Abort-1.       A  ->  T   :   S-SIG(A,[C,A,B,abort])
Abort-2.       T  ->  A   :   if (resolved) then S-SIG(B,C) else S-SIG(T,S-SIG(A,
Resolve-A-1.   A  ->  T   :   [PCS(B,C,A,T),S-SIG(A,C)]
Resolve-A-2.   T  ->  A   :   if (aborted) then S-SIG(T,S-SIG(A,[C,A,B,abort])) el
Resolve-B-1.   B  ->  T   :   [PCS(A,C,B,T),S-SIG(B,C)]
Resolve-B-2.   T  ->  B   :   if (aborted) then S-SIG(T,S-SIG(A,[C,A,B,abort])) el
```

## Description of the protocol rules

About cryptographic primitives involved :

- `S-SIG(X,M)` denotes standard signature of contractual text `M` by principal `A`,
- `PCS(A,M,B,T)` denotes private contract signature of contractual text `M` by `A` inside a session involving participant `B` and TTP `T`. It is assumed that such a construction has the following properties:

  - a "fake-version" of `PCS(A,M,B,T)` can be computed by `B`, identical to the true one from the point of view of an external observer `O` (distinct from `A`, `B` and `T`),
  - `PCS(A,M,B,T)` can be converted by `T` into a "TTP-signature", denoted `TTP-SIG(A,M)` and identical to `S-SIG(A,M)` from the point of view of an external observer.

About the execution of the protocol:

- when no problem appears between `A` and `B`, the `Exchange` subprotocol is able to complete contract distribution,
- after sending the first message, if `A` does not receive any response from `B`, she can run the `Abort` subprotocol,
- after sending the second message, if `B` does not receive any response from `A`, she can run the `Resolve-B` subprotocol,
- after sending the third message, if `A` does not receive any response from `B`, she can run the `Resolve-A` subprotocol.

## Requirements

This protocol was designed in order to satisfy the following properties:

- *completeness*: an adversary (submitted to some restrictions) cannot prevent two honest participants from obtaining a valid signature on a contractual text,
- *fairness*: it is impossible for a corrupted participant to obtain a valid contract without allowing the remaining participant to do the same. Moreover, once an honest participant has obtained an abort confirmation from the TTP, it is impossible for any other participant to obtain a valid contract. Finally, every honest participant is able to complete the protocol.
- *abuse-freeness*: it is impossible for a (possible corrupted) participant, at any point of the protocol, to be able to prove to an external observer that he has the power to determine the outcome of the protocol (validate or invalidate the contract).

## References

[GJM99]

### Claimed proofs

[SM01] [KR02] [CKS01]

### Claimed attacks

[SM01]

## Gong

**Author(s):** Li Gong 1989
*Last modified February 6, 2003*

**Summary:** Mutual authentication protocol of two principals with a trusted server, and exchange of a new symmetric key. Uses one-way functions and no encryption.

### Protocol specification (in common syntax)

```
A, B, S :      principal
Na, Nb, Ns :   number
Pa, Pb :       number
K, Ha, Hb :    number
f1 :           number, number, number, number -> number
f2 :           number, number, number, number -> number
f3 :           number, number, number, number -> number
g :            number, number, number, number -> number
xor :          number,number -> number
```

alias K = `f1(Ns,Na,B,Pa)`
alias Ha = `f2(Ns,Na,B,Pa)`
alias Hb = `f3(Ns,Na,B,Pa)`

```
1.   A  ->  B  :   A, B, Na
2.   B  ->  S  :   A, B, Na, Nb
3.   S  ->  B  :   Ns, xor(f1(Ns, Nb, A, Pb), K),
                   xor(f2(Ns, Nb, A, Pb), Ha),
                   xor(f3(Ns, Nb, A, Pb), Hb),
                   g(K, Ha, Hb, Pb)
4.   B  ->  A  :   Ns, Hb
5.   A  ->  B  :   Ha
```

### Description of the protocol rules

`f1`, `f2`, `f3`, and `g` are one-way functions.

Initially, the principal `A` (resp. `B`) shares the long-term secret `Pa` (resp. `Pb`) with the server `S`.

`Na`, `Nb` and `Ns` are nonces and `Ha`, `Hb` and `K` are just aliases for resp. `f2(Ns,Na,B,Pa)`, `f3(Ns,Na,B,Pa)`, and `f1(Ns,Na,B,Pa)`.

We assume commutativity and associativity for the `xor` operator,

```
xor(x, y) = xor(y, x)
xor(x, xor(y, z)) = xor(xor(x, y), z)
```

as well as the xor axioms:

```
xor(x, 0) = x
xor(x, x) = 0
```

Hence, the principal `B` can extract `Ha`, `Hb` and `K` from the message `3`, and check them using the checksum `g(K, Ha, Hb, Pb)`.

Knowing `Pa`, the principal `A` can construct `Ha`, `Hb` and `K` once he has received `Ns` in message `4`, he can check the check value for `Hb` he has just computed against the one sent by `B` in message `4` and send the last message.

### Requirements

The protocol must guaranty the secrecy of `K`: in every session, the value of `K` must be known only by the participants playing the roles of `A`, `B` and `S` in that session.

The protocol must also ensure mutual authentication of `A` and `B`.

### References

[Gon89]

## Kao Chow Authentication v.1

**Author(s):** I Long Kao and Randy Chow 1995
*Last modified November 11, 2002*

**Summary:** Key distribution and authentication protocol. Symmetric keys cryptography with server.

## Protocol specification (in common syntax)

```
A, B, S :        principal
Na, Nb :         number
Kab, Kbs, Kas :  key

1.   A  ->  S  :   A, B, Na
2.   S  ->  B  :   {A, B, Na, Kab}Kas, {A, B, Na, Kab}Kbs
3.   B  ->  A  :   {A, B, Na, Kab}Kas, {Na}Kab, Nb
4.   A  ->  B  :   {Nb}Kab
```

## Description of the protocol rules

Kas and Kbs are symmetric keys whose values are initially known only by A and S, respectively B and S.

Na and Nb are nonces for mutual authentication and to verify the authenticity of the fresh symmetric key Kab.

The messages 3 and 4 are *repeated authentication*: after that messages 1 and 2 have completed successfully, 3 and 4 can be played several times by B before starting a secrete communication with A encrypted with the session key kab (see also Neumann Stubblebine for repeated authentication).

## Remark

This protocol has been designed to prevent the freshness attack on the repeated authentication part of the Neumann Stubblebine protocol. Indeed, the nonce Na in the ciphers of message 2 prevent a shared key compromised after another run of the protocol to be reused.

However, as shown below, an attack of this kind is still possible. This flaw is fixed in Kao Chow Authentication v.2.

## Requirements

The protocol must guaranty the secrecy of Kab: in every session, the value of Kab must be known only by the participants playing the roles of A, B and S.

When A, resp. B, receives the key Kab in message 3, resp. 2, this key must have been issued in the same session by the server `S` with whom `A` has started to communicate in message `1`.

The protocol must also ensures mutual authentication of `A` and `B`.

### References

[KC95]. The protocol is presented as specified in [CJ97].

### Claimed attacks

As described in [KC95], this protocol suffers the same kind of attack as the Denning Sacco freshness attack on Needham Schroeder Symmetric Key, when an older session symmetric key `Kab` has been compromised.

```
i.1.      A    ->   S    :   A, B, Na
i.2.      S    ->   B    :   {A, B, Na, Kab}Kas, {A, B, Na, Kab}Kbs
                             assume that Kab is compromised
ii.1.                        Omitted
ii.2.   I(S)   ->   B    :   {A, B, Na, Kab}Kas, {A, B, Na, Kab}Kbs
ii.3.     B    ->  I(A)  :   {A, B, Na, Kab}Kas, {Na}Kab, N'b
ii.4.   I(A)   ->   B    :   {N'b}Kab
```

### See also

Kao Chow Authentication v.2, Kao Chow Authentication v.3, Needham Schroeder Symmetric Key, Neumann Stubblebine.

## Kao Chow Authentication v.2

**Author(s):** I Long Kao and Randy Chow 1995
*Last modified November 11, 2002*

**Summary:** Key distribution and authentication protocol. Symmetric keys cryptography with server.

### Remark

This protocol is a correction of Kao Chow Authentication v.1 to prevent a freshness attack à la Denning Sacco attack (see Needham Schroeder Symmetric Key).

### Protocol specification (in common syntax)

```
A, B, S :         principal
Na, Nb :          number
Kab, Kbs, Kas :   key

1.   A  ->  S  :   A, B, Na
2.   S  ->  B  :   {A, B, Na, Kab, Kt}Kas, {A, B, Na, Kab, Kt}Kbs
3.   B  ->  A  :   B, {A, B, Na, Kab, Kt}Kas, {Na, Kab}Kt, Nb
4.   A  ->  B  :   {Nb, Kab}Kt
```

### Description of the protocol rules

See Kao Chow Authentication v.1. `Kt` is an additional fresh symmetric key whose purpose is to prevent a freshness attack as in Kao Chow Authentication v.1.

### Requirements

See Kao Chow Authentication v.1.

### References

[KC95].

This specification of the protocol differs from the one in [CJ97].

### Claimed proofs

[KC95]

### See also

Kao Chow Authentication v.1, Kao Chow Authentication v.3, Needham Schroeder Symmetric Key, Neumann Stubblebine.

## Kao Chow Authentication v.3

**Author(s):** I Long Kao and Randy Chow 1995
*Last modified November 11, 2002*

**Summary:** Key distribution and authentication protocol. Symmetric keys cryptography with server.

## Remark

This protocol is an extension of Kao Chow Authentication v.2 to encompass tickets.

## Protocol specification (in common syntax)

| | |
|---|---|
| `A, B, S :` | `principal` |
| `Na, Nb :` | `number` |
| `Kab, Kbs, Kas :` | `key` |

```
1.   A  ->  S  :   A, B, Na
2.   S  ->  B  :   {A, B, Na, Kab, Kt}Kas, {A, B, Na, Kab, Kt}Kbs
3.   B  ->  A  :   {A, B, Na, Kab, Kt}Kas, {Na, Kab}Kt, Nb, {A, B, Ta, Kab}Kbs
4.   A  ->  B  :   {Nb, Kab}Kt, {A, B, Ta, Kab}Kbs
```

## Description of the protocol rules

In message `3`, `B` generates a new ticket `{A, B, Ta, Kab}Kbs` containing `Kab` and a timestamp `Ta`.

## Requirements

See Kao Chow Authentication v.1.

## References

[KC95].

## See also

Kao Chow Authentication v.1, Kao Chow Authentication v.2, Needham Schroeder Symmetric Key, Neumann Stubblebine.

# Kerberos V5

**Author(s):** B. Clifford Neuman and Theodore Ts'o 1994

*Last modified November 7, 2002*

**Summary:** Distribution of a symmetric key (in a *ticket*), for communication between a client and a server, with authentication.

## Remark

This protocol is based on based on the Needham Schroeder Symmetric Key protocol and uses timestamps and nonces to correct the flaw of Denning Sacco.

## Protocol specification (in common syntax)

```
A, G, C, S, U :           principal
N1, N2 :                  nonce
L1, L2 :                  nonce
T1start, T1expire :       timestamp
T2start, T2expire :       timestamp
Kcg, Kcs, Kag, Ku, Kgs :  key

1.   C  ->  A  :   U, G, L1, N1
2.   A  ->  C  :   U, {U, C, G, Kcg, T1start, T1expire}Kag,
                   {G, Kcg, T1start, T1expire}Ku
3.   C  ->  G  :   S, L2, N2, {U, C, G, Kcg, T1start, T1expire}Kag,
                   {C, T1}Kcg
4.   G  ->  C  :   U, {U, C, S, Kcs, T2start, T2expire}Kgs,
                   {S, Kcs, T2start, T2expire, N2}Kcg
5.   C  ->  S  :   {U, C, S, Kcs, T2start, T2expire}Kgs,
                   {C, T2}Kcs
6.   S  ->  C  :   {T2}Kcs
```

## Description of the protocol rules

`C` is a client,

`S` is a a server (`C` wants to communicate with `S`),

`U` is a user on behalf of which `A` and `S` communicate,

`G` is a ticket granting server,

`A` is a key distribution center (trusted server).

The keys Kag and Kgs are long term symmetric key whose values are supposed to be known initially only by, A and G, respectively G and S.

L1 and L2 are lifetimes, N1 and N2 are nonces. T1start, T1expire, T2start, T2expire are time stamps which define the interval of validity of the ticket in which they are contained.

U is a user on behalf of whom the client C communicates. In particular, C initially knows the value of the key Ku.

The key Kcg is freshly generated by A for communication between C and G, and in transmitted to C in message 2, encrypted by Ku, and indirectly to G, in the *ticket* {U, C, G, Kcg, T1start, T1expire}Kag which C transmits blindly to G in message 3.

The authentificator {C, T1}Kcg is used by G to check timeliness of the ticket.

The key Kcs is freshly generated by G for communication between C and G, and in transmitted to C in message 4, encrypted by Kcg, and indirectly to S, in the *ticket* {U, C, S, Kcs, T2start, T2expire}Kgs which C transmits blindly to S in message 5.

## Requirements

The protocol must guaranty the secrecy of Kcs: in every session, the value of K must be known only by the participants playing the roles of A, B and S in that session.

A and C must agree on the values of T1start and T1expire.

G and C must agree on the values of T2start and T2expire and T1.

C and S must agree on the value of T2.

## References

[NT94]

## Claimed proofs

- [NT94]
- [BAN89]
- [SMB90] modelization with Abstract State Machines (stepwise refinements), and (manual) proof of correctness.

### See also

Needham Schroeder Symmetric Key

## KSL

**Author(s):** Axel Kehne and Jürgen Schönwälder and Horst Langendörfer
1992
*Last modified December 2, 2002*

**Summary:** Nonce based improvement of Kerberos V5 protocol with generalized timestamps. Distribution of a session key and a ticket and repeated mutual authentication. Symmetric key cryptography with server.

### Protocol specification (in common syntax)

```
A, B, S :             principal
Na, Nb, Nc, Ma, Mb :  number
Kas, Kbs, Kab, Kbb :  key
Tb :                  generalizedTimestamp

1.   A  ->  B  :   Na, A
2.   B  ->  S  :   Na, A, Nb, B
3.   S  ->  B  :   {Nb, A, Kab}Kbs, {Na, B, Kab}Kas
4.   B  ->  A  :   {Na, B, Kab}Kas, {Tb, A, Kab}Kbb, Nc, {Na}Kab
5.   A  ->  B  :   {Nc}Kab

6.   A  ->  B  :   Ma, {Tb, A, Kab}Kbb
7.   B  ->  A  :   Mb, {Ma}Kab
8.   A  ->  B  :   {Mb}Kab
```

### Description of the protocol rules

The messages `1-5` are the part concerning the generation and exchange of the session key `Kab`. The messages `6-8` are for mutual authentification. This second part of the protocol is also called *repeated authentication* because it can be repeated alone several times, until the ticket `{Tb, A, Kab}Kbb` expires.

**Key exchange.** The keys `Kas` and `Kbs` are long term symmetric key whose values are supposed to be known initially only by `A` and `S`, respectively `B` and S.

The session key Kab is freshly generated by S and in sent in message 3 directly to B, and indirectly to A, in the cipher {Na, B, Kab}Kas, transmitted blindly to A by B in message 4.

Kbb is a secret key only known to B, used to encrypt the ticket {Tb, A, Kab}Kbb in message 4. This ticket will be used in the repeated authentication.

**Repeated authentication.** In the ticket {Tb, A, Kab}Kbb, Tb is a generalized timestamp, made of a timestamp from the local clock of B, a lifetime limiting the validity of the ticket (relatively to the local clock of B) and a clock identifier, i.e. a nonce which is updated each time B's local clock is corrected.

When he receives a ticket in message 6, B compares the time identifier in Tb to the current identifier of his local clock and if they match, verifies the validity of the ticket, i.e. he checks that the time of his local clock is within the time window defines by the timestamp and the lifetime of Tb. If one of these tests fails, then B rejects the ticket. Otherwise, he starts an exchange of nonces (messages 7 and 8) the purpose of which is to convince mutually A and B that they both possess the session key Kab.

## Requirements

The protocol must guaranty the secrecy of Kab: in every session, the value of Kab must be known only by the participants playing the roles of A, B and S in that session.

The protocol must also ensures mutual authentication of A and B.

## References

[KSL92]

## Claimed proofs

The authors of the protocol propose in [KSL92] an analysis in the framework of the BAN logic [BAN89].

## Claimed attacks

**1.** [Low96]: “The repeated authentification part can be used as an encrypting oracle”. If the intruder `I` wants to encrypt some data `M` with the session key `Kab`, he can run:

```
6.    I(A)  ->   B    :   M, {Tb, A, Kab}Kbb
7.      B   -> I(A)   :   Mb, {M}Kab
```

The ticket `{Tb, A, Kab}Kbb` can have been learned by `I` from the message `4` of a previous key distribution. After running the two above message, `I` can send:

```
I(A)  ->  B   :     {M}Kab
```

and `B` will accept this message as having been sent by `A`.

**2.** The attacks presented in [HLL+95] on the repeated authentication part of the `neumannStubblebine` protocol also works here.

This attack concerns the repeated authentication part, assuming `Kab` has been recorded in a previous legitimate run of the protocol.

```
i.6.     I(A)  ->   B    :   Mi, { Tb, A, Kab}Kbb
i.7.       B   -> I(A)   :   Mb, {Mi}Kab
ii.6.    I(A)  ->   B    :   Mb, {Tb, A, Kab}Kbb
ii.7.      B   -> I(A)   :   Mb', {Mb}Kab
i.8.     I(A)  ->   B    :   {Mb}Kab
```

**3.** [Low96]. In this scenario, two tickets generated by two different agents contains the same session key `Kab`, which, according to [Low96], was supposed not to happen in the protocol of [KSL92].

```
i.1.     I(A)  ->   B    :   Ni, A
i.2.       B   -> I(S)   :   Ni, A, Nb, B
ii.1.    I(B)  ->   A    :   Nb, B
ii.2.      A   ->   S    :   Nb, B, Na, A
ii.3.      S   ->   A    :   {Na, B, Kab}Kas, {Nb, A, Kab}Kbs
ii.4.      A   -> I(B)   :   {Nb, A, Kab}Kbs, {Ta, B, Kab}Kaa, Nc, {Nb}Kab
i.3.     I(S)  ->   B    :   {Nb, A, Kab}Kbs, {Na, B, Kab}Kas
i.4.       B   -> I(A)   :   {Na, B, Kab}Kas, {Tb, A, Kab}Kbb, Nc', {Ni}Kab
```

The intruder `I` can then use the two tickets to complete step `5` of both runs `i` and `ii`. In this scenarion, the repeated authentication procedure is used as an encrypting oracle.

```
i.6.    I(A)  ->   B    :   Nc, {Tb, A, Kab}Kbb
i.7.     B    ->  I(A)  :   Mb, {Nc}Kab
ii.5.   I(B)  ->   A    :   {Nc}Kab
ii.6.   I(B)  ->   A    :   Nc', {Ta, B, Kab}Kaa
ii.7.    A    ->  I(B)  :   Ma, {Nc'}Kab
i.5.    I(A)  ->   B    :   {Nc'}Kab
```

**4.** [Low96]. The ticket obtained in the first part of the above scenario also permits I to impersonate A in the repeated authentification part of the protocol.

```
i.6.    I(A)  ->   B    :   Mi, {Tb, A, Kab}Kbb
i.7.     B    ->  I(A)  :   Mb, {Mi}Kab
ii.6.   I(B)  ->   A    :   Mb, {Ta, B, Kab}Kaa
ii.7.    A    ->  I(B)  :   Ma, {Mb}Kab
i.8.    I(A)  ->   B    :   {Mb}Kab
```

### See also

Kerberos V5, Neumann Stubblebine, Lowe modified KSL.

## Lowe modified KSL

**Author(s):** Gavin Lowe 1996
*Last modified December 2, 2002*

**Summary:** Lowe modified version of the KSL protocol to prevent authentication attacks. Distribution of a session key and a ticket and repeated mutual authentication. Symmetric key cryptography with server.

### Protocol specification (in common syntax)

```
A, B, S :             principal
Na, Nb, Nc, Ma, Mb :  number
Kas, Kbs, Kab, Kbb :  key
Tb :                  generalizedTimestamp
```

```
1.   A  ->  B  :   Na, A
2.   B  ->  S  :   Na, A, Nb, B
3.   S  ->  B  :   {A, Nb Kab}Kbs, {Na, B, Kab}Kas
4.   B  ->  A  :   {Na, B, Kab}Kas, {Tb, A, Kab}Kbb, Nc, {B, Na}Kab
5.   A  ->  B  :   {Nc}Kab

6.   A  ->  B  :   Ma, {Tb, A, Kab}Kbb
7.   B  ->  A  :   Mb, {Ma, B}Kab
8.   A  ->  B  :   {A, Mb}Kab
```

### Description of the protocol rules

See KSL.

The version given above has been devised from the following modifications to KSL advised in [Low96]:

- “change the message 3 so that the two encrypted components have different forms” (in order to break symmetry),
- “make the encrypted components in 4, 7 and 8 different”,
- “include either A’s or B’s identity in these last three components”.

### Requirements

See KSL.

### References

[Low96]

### See also

KSL

## Neumann Stubblebine

**Author(s):** B. Clifford Neumann and Stuart G. Stubblebine April 1993
*Last modified November 8, 2002*

**Summary:** Session key exchange inspired by the Yahalom protocol with the addition of timestamps, and mutual authentication. Symmetric key cryptography with server.

## Protocol specification (in common syntax)

```
A, B, S :          principal
Na, Ma, Nb, Mb :   number
Kas, Kbs, Kab :    key
Ta, Tb :           time

1.   A  ->  B  :   A, Na
2.   B  ->  S  :   B, {A, Na, Tb}Kbs, Nb
3.   S  ->  A  :   {B, Na, Kab, Tb}Kas, {A, Kab, Tb}Kbs, Nb
4.   A  ->  B  :   {A, Kab, Tb}Kbs, {Nb}Kab
5.   A  ->  B  :   Ma, {A, Kab, Tb}Kbs
6.   B  ->  A  :   Mb, {Ma}Kab
7.   A  ->  B  :   {Mb}Kab
```

## Description of the protocol rules

The messages `1-4` are the part concerning the generation and exchange of the session key `Kab`. The messages `5-7` are the mutual authentification, this second part of the protocol can be repeated alone several times, until the ticket `{A, Kab, Tb}Kbs` expires (it is called *repeated authentication*).

## Requirements

The protocol must guaranty the secrecy of `Kab`: in every session, the value of `Kab` must be known only by the participants playing the roles of `A`, `B` and `S` in that session.

The protocol must also ensures mutual authentication of `A` and `B`.

## References

[NS93]

## Claimed attacks

**1.** From [HLL+95], see also BAN simplified version of Yahalom for the first 4 messages, where `B` accepts the nonce `Na` has the fresh shared key `Kab`.

```
 1.    I(A)  ->   B    :    A, Na
 2.     B    ->  I(S)  :    B, {A, Na, Tb}Kbs, Nb
 3.                         omitted
 4.    I(A)  ->   B    :    {A, Na, Tb}Kbs, {Nb}Na
 5.    I(A)  ->   B    :    Ma, {A, Na, Tb}Kbs
 6.     B    ->  I(A)  :    Mb, {Ma}Na
 7.    I(A)  ->   B    :    {Mb}Na
```

See Hwang modified version of Neumann Stubblebine for a modified version preventing this attack.

**2.** From [HLL+95]. This attack concerns the repeated authentication part, assuming Kab has been recorded in a previous legitimate run of the protocol.

```
 i.5.     I(A)  ->   B    :    Ma, { A, Kab, Tb }Kbs
 i.6.      B    ->  I(A)  :    Mb, {Ma}Kab
 ii.5.    I(A)  ->   B    :    Mb, {A, Kab, Tb}Kbs
 ii.6.     B    ->  I(A)  :    Mb', {Mb}Kab
 i.7.     I(A)  ->   B    :    {Mb}Kab
```

**3.** From [Wei99]. In this attack, the intruder I can get as many ciphers {A, Kiab, Tb}Kbs as needed to start a known plaintext attack in order to break Kbs.

```
 a.2.    I(B)  ->   S    :    B, {A, K0ab, Tb}Kbs, Nb
 a.3.     S    ->  I(A)  :    {B, Na, K1ab, Tb}Kas, {A, K1ab, Tb}Kbs, Nb
 b.2.    I(B)  ->   S    :    B, {A, K1ab, Tb}Kbs, Nb
 b.3.     S    ->  I(A)  :    {B, Na, K2ab, Tb}Kas, {A, K2ab, Tb}Kbs, Nb
                              etc
```

### See also

Yahalom

## Hwang modified version of Neumann Stubblebine

**Author(s):** T. Hwang and N.Y. Lee and C.M. Li and M.Y. Ko and Y.H. Chen 1995
*Last modified November 11, 2002*

**Summary:** Modified version of the Neumann Stubblebine protocol, to correct attack of the repeated authentification part.

## Protocol specification (in common syntax)

```
A, B, S :          principal
Na, Ma, Nb, Mb :   number
Kas, Kbs, Kab :    key
Ta, Tb :           time

1.    A  ->  B   :    A, Na
2.    B  ->  S   :    B, {A, Na, Tb, Nb}Kbs
3.    S  ->  A   :    {B, Na, Kab, Tb}Kas, {A, Kab, Tb}Kbs, Nb
4.    A  ->  B   :    {A, Kab, Tb}Kbs, {Nb}Kab
5.    A  ->  B   :    Ma, {A, Kab, Tb}Kbs
6.    B  ->  A   :    Mb, {Mb}Kab
7.    A  ->  B   :    {Mb}Kab
```

## Description of the protocol rules

See Neumann Stubblebine. The messages `1-4` are the part concerning the generation and exchange of the session key `Kab`. The messages `5-7` is the *repeated authentication* part.

## Requirements

See Neumann Stubblebine.

## References

[HLL+95]

## Claimed attacks

In [CJ95], Clark and Jacob describe an attack depending on the encryption method (when cipher block chaining is performed for encryption).

## See also

Neumann Stubblebine

## Needham-Schroeder Public Key

**Author(s):** Roger Needham and Michael Schroeder December 1978
*Submitted by Ralf Treinen November 4, 2002*
*Last modified December 9, 2002*

**Summary:** Mutual authentication, using a trusted key server and public keys.

### Protocol specification (in common syntax)

```
A,B,S :                         Principal
Na,Nb :                         Nonce
KPa,KPb,KPs,KSa,KSb,KSs :       Key
KPa,KSa :                       is a key pair
KPb,KSb :                       is a key pair
KPs,KSs :                       is a key pair

1.    A  ->  S   :    A,B
2.    S  ->  A   :    {KPb, B}KSs
3.    A  ->  B   :    {Na, A}KPb
4.    B  ->  S   :    B,A
5.    S  ->  B   :    {KPa, A}KSs
6.    B  ->  A   :    {Na, Nb}KPa
7.    A  ->  B   :    {Nb}KPb
```

### Description of the protocol rules

This protocol has been proposed by [NS78b]. In this protocol description, `KSa` (resp. `KSb`, `KSs`) is the secret key corresponding to the public key `KPa` (resp. `KPb`, `KPs`).

### Requirements

After completion of the protocol, the two principals `A` and `B` should be convinced about the identity of their respective correspondent.

### References

[NS78b].

### Claimed proofs

Burrows, Abadi and Needham [?] prove the correctness of the protocol in the sense of their logical framework. However, they point out a possible replay attack which, according to them, could be avoided by using timestamps.

### Claimed attacks

An intruder `I` may impersonate `A`, by inciting `A` to initiate a second session[Low95]. In the following, we ignore the message exchanges with the public key server and only consider messages between the principals `A` and `B`, and the intruder `I`. We assume that the intruder `I` possesses a key pair `(KPi, KSi)`, and we may also assume that every principal knows the public keys `KPa`, `KPb` and `KPi`.

```
 i.3.      A    ->   I    :   {Na,A}KPi
 ii.3.   I(A)   ->   B    :   {Na,A}KPb
 ii.6.     B    ->  I(A)  :   {Na,Nb}KPa
 i.6.      I    ->   A    :   {Na,Nb}KPa
 i.7.      A    ->   I    :   {Nb}KPi
 ii.7.   I(A)   ->   B    :   {Nb}KPb
```

### Remark

It has been proposed to fix the protocol by including the respondent's identity in the response [Low95].

### See also

Lowe's fixed version of Needham-Schroder Public Key

## Lowe's fixed version of Needham-Schroder Public Key

**Author(s):** Gavin Lowe 1995
*Last modified November 6, 2002*

**Summary:** This protocol is an amended version of the Needham-Schroeder Public Key. Its purpose id mutual authentication, using a trusted keyserver and public keys.

## Protocol specification (in common syntax)

```
A,B,S :                      Principal
Na,Nb :                      Nonce
KPa,KPb,KPs,KSa,KSb,KSs :    Key
KPa,KSa :                    is a key pair
KPb,KSb :                    is a key pair
KPs,KSs :                    is a key pair

1.   A  ->  S   :   A,B
2.   S  ->  A   :   {KPb, B}KSs
3.   A  ->  B   :   {Na, A}KPb
4.   B  ->  S   :   B,A
5.   S  ->  B   :   {KPa, A}KSs
6.   B  ->  A   :   {Na, Nb, B}KPa
7.   A  ->  B   :   {Nb}KPb
```

## Description of the protocol rules

Compared to the original version of the Needham-Schroeder Public Key protocol, the identity of the responder `B` has been added in the message `6` to prevent the attack discovered in [Low95].

## Requirements

See Needham-Schroeder Public Key.

## References

[Low95]

## Claimed proofs

It is reported in [Low95] that the technique that permitted to find the Lowe attack on the Needham-Schroeder Public Key protocol (running FDR on a CSP presentation of the protocol) found no attack on this protocol.

## See also

Needham-Schroeder Public Key

## Needham Schroeder Symmetric Key

**Author(s):** Roger Needham and Michael Schroeder 1978
*Last modified November 8, 2002*

**Summary:** Distribution of a shared symmetric key by a trusted server and mutual authentification. Symmetric key cryptography with server.

### Protocol specification (in common syntax)

```
A, B, S :          principal
Na, Nb :           nonce
Kas, Kbs, Kab :    key
dec :              nonce -> nonce

1.    A  ->  S   :    A, B, Na
2.    S  ->  A   :    {Na, B, Kab, {Kab, A}Kbs}Kas
3.    A  ->  B   :    {Kab,A}Kbs
4.    B  ->  A   :    {Nb}Kab
5.    A  ->  B   :    {dec(Nb)}Kab
```

### Description of the protocol rules

This protocol establishes the fresh shared symmetric key `Kab`.

Messages `1-3` perform the distribution of the fresh shared symmetric key `Kab` and messages `4-5` are for mutual authentification of `A` and `B`.

The operator `dec` is decrementation.

### Requirements

The protocol must guaranty the secrecy of `Kab`: in every session, the value of `Kab` must be known only by the participants playing the roles of `A`, `B` and `S` in that session.

If the participant playing `B` accepts the last message `5`, then `Kab` has been sent in message `3.` by `A` (whose identity is included in the cipher of message `3`).

### References

[NS78a].

### Claimed attacks

Authentication attack by Denning and Sacco [DS81]. Assume that `I` has recorded the session `i` and that `Kab` is compromised. After the session `ii`, `B` is convinced that he shares the secret key `Kab` only with `A`.

```
i.1.      A    ->    S     :   A, B, Na
i.2.      S    ->    A     :   {Na, B, Kab, {Kab, A}Kbs }Kas
i.3.      A    ->   I(B)   :   {Kab,A}Kbs
                               assume that Kab is compromised
ii.3.    I(A)  ->    B     :   {Kab,A}Kbs
ii.4.     B    ->   I(A)   :   {Nb}Kab
ii.5.    I(A)  ->    B     :   {dec(Nb)}Kab
```

### See also

Amended Needham Schroeder Symmetric Key, Denning-Sacco shared key, Kerberos V5.

## Amended Needham Schroeder Symmetric Key

**Author(s):** Roger Needham and Michael Schroeder January 1987
*Last modified November 8, 2002*

**Summary:** This is an amended version of Needham Schroeder Symmetric Key, by the same authors. Distribution of a shared symmetric key by a trusted server and mutual authentification. Symmetric key cryptography with server.

### Protocol specification (in common syntax)

```
A, B, S :          principal
Na, Nb :           number
Kas, Kbs, Kab :    key
dec :              number -> number

1.    A  ->  B  :   A
2.    B  ->  A  :   {A, Nb}Kbs
3.    A  ->  S  :   A, B, Na, {A, Nb}Kbs
4.    S  ->  A  :   {Na, B, Kab, {Kab, Nb, A}Kbs}Kas
5.    A  ->  B  :   {Kab, Nb, A}Kbs
6.    B  ->  A  :   {Nb}Kab
7.    A  ->  B  :   {dec(Nb)}Kab
```

### Description of the protocol rules

See Needham Schroeder Symmetric Key. The extra exchange of the nonce `Nb` prevents the Denning Sacco freshness attack described in Needham Schroeder Symmetric Key.

### Requirements

See Needham Schroeder Symmetric Key.

### References

[NS87].

### See also

Needham Schroeder Symmetric Key, Kerberos V5.

## Otway Rees

**Author(s):** D. Otway and O. Rees January 1997
*Last modified November 12, 2002*

**Summary:** Distribution of a shared symmetric key by a trusted server. Symmetric key cryptography with server.

### Protocol specification (in common syntax)

```
A, B, S :         principal
M, Na, Nb :       nonce
Kas, Kbs, Kab :   key

1.   A  ->  B  :   M, A, B, {Na, M, A, B}Kas
2.   B  ->  S  :   M, A, B, {Na, M, A, B}Kas , {Nb, M, A, B}Kbs
3.   S  ->  B  :   M, {Na, Kab}Kas, {Nb, Kab}Kbs
4.   B  ->  A  :   M, {Na, Kab}Kas
```

### Description of the protocol rules

The nonce `M` identifies the session number.

Kas and Kbs are symmetric keys whose values are initially known only by A and S, respectively B and S.

Kab is a fresh symmetric key generated by S in message 3 and distributed to B, directly in message 3, and to A, indirectly, when B forwards blindly {Na, Kab}Kas to A in message 4.

### Requirements

The protocol must guaranty the secrecy of Kab: in every session, the value of Kab must be known only by the participants playing the roles of A, B and S.

When A, resp. B, receives the key Kab in message 3, resp. 2, this key must have been issued in the same session by the server S with whom B has started to communicate in message 2.

### References

[OR87]

### Claimed attacks

Type flaw in [CJ97], where A will accept in last message 4 the triple (M, A, B) as a fresh key Kab.

```
1.     A    ->  I(B)   :   M, A, B, {Na, M, A, B}Kas
2.     B    ->   S     :   M, A, B, {Na, M, A, B}Kas , {Nb, M, A, B}Kbs
3.     S    ->   B     :   M, {Na, Kab}Kas, {Nb, Kab}Kbs
4.   I(B)   ->   A     :   M, {Na, M, A, B}Kas
```

## Schnorr’s Protocol

**Author(s):** C. P. Schnorr 1991

**Summary:** The Schnorr protocol is described by R. Cramer, I. Damgård and B. Schoenmakers in [CDS94].

## Protocol specification (in common syntax)

```
A, B :             principal
Na, Nb :           fresh number
Sa :               private key
Pa = exp(g,Sa) :   public key
```

```
A chooses Na and computes a = exp(g,Na)
 1.   A  ->  B  :    a
B chooses Nb
 2.   B  ->  A  :    Nb
A computes r = Na + Nb × Sa
 3.   A  ->  B  :    r
B checks that exp(g, r) = a × exp(Pa,Nb)
```

## Description of the protocol rules

A zero-knowledge protocol is designed for convincing the verifier of the validity of a given statement, without releasing any knowledge beyond the validity of the statement. This concept was introduced in [GMR85]. An overview can be found in [Gol01]. We present the Schnorr protocol which is described by R. Cramer, I. Damgård and B. Schoenmakers in [CDS94] and which uses this method.

The `+`, × and `exp` symbols denote respectively addition, multiplication and modular exponentiation.

Details of the computation done by `B` at the last step of the protocol:

```
    a × exp(Pa,Nb)
 =  exp(g,Na) × exp(exp(g,Sa),Nb)
 =  exp(g,Na) × exp(g,Sa × Nb)
 =  exp(g,Na + Sa × Nb)
 =  exp(g, r)
```

## Requirements

`A` wants to prove his identity to `B` by showing him that he knows `Sa` without revealing it.

### References

[CDS94]

## Shamir-Rivest-Adleman Three Pass Protocol

**Author(s):** A. Shamir, R. Rivest, L. Adleman

**Summary:** The following protocol, described in [CJ97], allows two principals to exchange a secret message without sharing any initial secret.

### Protocol specification (in common syntax)

```
A, B :    principal
Ka, Kb :  symkey
M :       fresh number

1.   A  ->  B  :   {M}Ka
2.   B  ->  A  :   {{M}Ka}Kb
3.   A  ->  B  :   {M}Kb
```

### Description of the protocol rules

This protocol assumes that encryption is commutative, *i.e.*

$$\{\{x\}y\}z = \{\{x\}z\}y.$$

The initiator A encrypts his message M by his secret key Ka, then B encrypts the message he received by his secret key Kb. Since {{M}Ka}Kb = {{M}Kb}Ka, the agent A can decr ypt it and send {M}Kb to B. Then, using Kb, B can retrieve M.

## SK3

**Author(s):** Victor Shoup and Avi D. Rubin 1996
*Last modified February 11, 2003*

**Summary:** Symmetric key distribution using Smart Cards, by Shoup and Rubin.

## Protocol specification (in common syntax)

```
A, B, S, Ca, Cb :  principal
Ka, Kb :           symkey
Kac, Kbc :         symkey
Na, Nb :           nonce
0,1,2 :            number
alias Kab = {A, 0}Kb
alias Pab = Kab + {B, 1}Ka

1.    A   ->  S   :  A, B
2.    S   ->  A   :  Pab, {Pab, B, 2}Ka
3.    A   ->  Ca  :  A
4.    Ca  ->  A   :  Na, {Na, 1, 1}Kac
5.    A   ->  B   :  A, Na
6.    B   ->  Cb  :  A, Na
7.    Cb  ->  B   :  Nb, {Nb, 0, 0}Kab, {Na, Nb, 1}Kab, {Nb, 0, 1}Kab
8.    B   ->  A   :  Nb, {Na, Nb, 1}Kab
9.    A   ->  Ca  :  B, Na, Nb, Pab, {Pab, B, 2}Ka, {Na, Nb, 1}Kab, {Nb, 0, 1}Ka
10.   Ca  ->  A   :  {Nb, 0, 0}Kab, {Nb, 0, 1}Kab
11.   A   ->  B   :  {Nb, 0, 1}Kab
```

## Description of the protocol rules

- the operator {M}K denotes DES encryption.
- the operator + is xor.
- the principal Ca (resp. Cb is a smart card connected to A (resp. B) and used to store its long term keys.
- **NB:** the connection between A and Ca (resp. B and Cb) is assumed to be secure (i.e. no intruder has the capability to listen to this connection).
- Ka (resp. Kb) is a long term (symmetric) keys associated to the principal A (resp. B). It is assumed to be known initially only by Ca (resp. Cb) and the server S.
- Kac (resp. Kbc) is a secret symmetric key share (and initially only known by) A and Ca (resp. B and Cb).
- 0, 1, 2 are arbitrary padding constants, known to every principal.

1,2 A requires and obtains from the server S the pair key Pab associed to A and B. {Pab, B, 2}Ka is a verifier for this value.

3,4 A requires and abtains a nonce Na from her smart card Ca. {Na, 1, 1}Kac is a verifier. In [SR96], it is suggested to use a 8 bytes counter on Ca to generate Na.

5 A sends the nonce, meaning she request the establishment of a session symmetric key.

6,7 B obtains the nonce Nb from Cb (same remark as in 3,4 for the counters). {Nb, 0, 0}Kb is a session key and {Na, Nb, 1}Kab, and {Nb, 0, 1}Kab are verifiers respectively for A and B.

8 B transmits the nonce Nb and A's verifier to A.

9 the nonce Nb and A's verifier are transmitted to A.

10 A's smart card Ca makes the verifications, computes the session key {Nb, 0, 0}Kb and transmits it to A.

11 A aknowledge to B, who can compare this message to his verifier remaining from message 7.

## Requirements

The session key {Nb, 0, 0}Kb must remain secret.

## References

[SR96]. Some variants and implementation issues are discussed in the update [Sho96]. See also the implementor's paper [JHC$^+$98].

## Claimed proofs

The proof of [SR96] is based on the probabilistic definition of secure key distribution from Bellare and Rogaway [BR95].

[Bel01] uses a theorem proving approach, following Paulson's inductive method.

## Remark

See [Sho96]. The nonce Na that A obtains from his smart card Ca must actually be truly random and not implemented by counters as first suggested in [SR96].

Indeed, if the next value of `Na` (sent in message `5` of session `i`) is predictable (let us call it `Na'`), then then the intruder `I` can query `B` for the verifiers including `Na'` (session `ii`) and use them to answer the next challenge of `A` (hence, authentication error in session `iii`).

```
i.5.       A    ->    B     :    A, Na
ii.5.    I(A)   ->    B     :    A, Na'
ii.6.      B    ->    Cb    :    A, Na'
ii.7.      Cb   ->    B     :    Nb', {Nb', 0, 0}Kab, {Na', Nb', 1}Kab, {Nb', 0, 1}Ka
ii.8.      B    ->    A     :    Nb', {Na', Nb', 1}Kab
iii.5.     A    ->   I(B)   :    A, Na'
iii.8.   I(B)   ->    A     :    Nb', {Na', Nb', 1}Kab
```

According to [Sho96], the nonce `Nb` may though be a counter.

## SmartRight view-only

**Author(s):** Jean-Pierre Andreaux, Sylvain Chevreau, Eric Diehl 2001
*Submitted by Thomas Genet March 6, 2003*
*Last modified March 6, 2003*

**Summary:** This *view-only* protocol is part of the SmartRight system designed by Thomson for copy protection for the Digital Video Broadcasting technology. Its purpose is to ensure that the digital content broadcasted can be view only once. It uses symmetric key cryptography with nonces and xor.

### Protocol specification (in common syntax)

```
CC, TC :                 principal
VoKey, VoR, VoRi, CW :   number
Kc :                     key
h :                      number -> number

1.   CC  ->  TC  :   {VoKey, CW+VoR}Kc
2.   TC  ->  CC  :   VoRi
3.   CC  ->  TC  :   VoR, {h(VoRi)}VoKey
```

### Description of the protocol rules

The above presentation and these explanations are extracted from [GTTT03]:

The protocol is deployed between two smartcards: `CC` (Converter Card) and `TC` (Terminal Card), respectively in an access device (i.e. a digital receiver) receiving a scrambled digital content and a presentation device (i.e.

a television) which is supposed to descramble the content before rendering it. The keys used to scramble the content are called *control words* `CW`.

The cards `CC` and `TC` share a secret symmetric encryption key `Kc`.

`CC` generates first the random values `VoKey` and `VoR`, and sends them encrypted to `TC` (in message `1`), together with a `CW` which has been received (by the access divice) with the scrambled content. The operator `+` is xor.

`TC` then sends in return a random challenge `VoRi` (message `2`). The message `3` is the answer of `CC` to the challenge. After receiving the message `3`, `TC` checks if the answer is correct, by comparing the hashed value `h(VoRi)` with its own value, and if so, it extracts `CW` and uses it to descramble the content.

**Note on memory management:**

- After sending the message `3`, `CC` deletes `VoR` and `VoKey` from its memory.
- After receiving and accepting the message `3`, `TC` deletes `VoRi` from its memory.

This is very important (with respect to the property below) because of the control aspects given below.

**Note on control:** The principal `CC` and `TC` switch to the next state only once they have received the next message expected (and not once they have send a message as usual). More precisely:

- After sending message `2`, `TC` will continue to accept a (new) message `1` and reply by a (new) `VoRi` (message `2`) until it receives the message `3` and accepts it.
- After sending message `2`, `TC` will process all the messages `3` it shall receive until it receives a new message `1`.

## Requirements

cited from [GTTT03]: The control word `CW` may be extracted by `TC` only once at the time where the protocol is played.

## References

[Tho01], [GTTT03].

### Claimed proofs

The above property is proved in [GTTT03] in an automated way on a term rewriting model using Timbuk, a verification tool using abstract interpretation over tree automata domains.

## SPLICE/AS

**Author(s):** Suguru Yamaguchi, Kiyohiko Okayama, and Hideo Miyahara
November 1991
*Last modified November 26, 2002*

**Summary:** Mutual authentication protocol. Public key cryptography with a certification authority signing and distributing public keys.

### Protocol specification (in common syntax)

```
S, C, AS :    principal
N1, N2, N3 :  nonce
T :           timestamp
L :           lifetime
pk, sk :      principal -> key (keypair)

1.   C   ->  AS  :   C, S, N1
2.   AS  ->  C   :   AS, {AS, C, N1, pk(S)}sk(AS)
3.   C   ->  S   :   C, S, {C, T, L, {N2}pk(S)}sk(C)
4.   S   ->  AS  :   S, C, N3
5.   AS  ->  S   :   AS, {AS, S, N3, pk(C)}sk(AS)
6.   S   ->  C   :   S, C, {S, inc(N2)}pk(C)
```

### Description of the protocol rules

`key` is the type of public/private keys. The functions `pk` and `sk` associate to a `principal`'s name its public key, resp. private key.

We assume that initially, the client `C` and the server `S` only know their own public and private key, and that the authority `AS` known the function `pk`, i.e. he knows everyone's public key.

`{AS, C, N1, pk(S)}sk(AS)` (in message 2) and `{AS, S, N3, pk(C)}sk(AS)` (in message 5) are certificates signed and distributed by the authority `AS`, for the respective public keys `pk(S)` and `pk(C)`.

After a successfull run of the protocol, the value of N2 can be used by C and S as a symmetric key for secure communications.

## Requirements

The protocol must guaranty the secrecy of N2: in every session, the value of N2 must be known only by the participants playing the roles of C, S.

The protocol must also ensure C that S has received N2 and S that the N2 he has received in message 3 originated from C.

## References

[YOM91]

## Claimed attacks

**1.** In an attack described in [HC95], the intruder I can impersonate the client C and obtain N2 in a single session (i.e. without even running a parallel session).

```
1.     I    ->   AS    :   I, S, N1
2.    AS    ->    I    :   AS, {AS, I, N1, pk(S)}sk(AS)
3.   I(C)   ->    S    :   C, S, {C, T, L, {N2}pk(S)}sk(I)
4.     S    ->  I(AS)  :   S, C, N3
4.   I(S)   ->   AS    :   S, I, N3
5.    AS    ->    S    :   AS, {AS, S, N3, pk(I)}sk(AS)
6.     S    ->  I(C)   :   S, C, {S, inc(N2)}pk(I)
```

In message 5, the server S accepts the certificate {AS, S, N3, pk(I)}sk(AS) from AS as a certificate of the public key of C (note that the certificates do not contain the name of the owner of public keyx in this protocol) and hence crypts the data in the last message 6 with the public key of I.

**2.** In this second (symmetric) attack from [HC95], the intruder I can impersonate the server S and obtain N2.

```
1.     C    ->  I(AS)  :   C, S, N1
1.   I(C)   ->   AS    :   C, I, N1
2.    AS    ->    C    :   AS, {AS, C, N1, pk(I)}sk(AS)
3.     C    ->  I(S)   :   C, S, {C, T, L, {N2}pk(I)}sk(C)
4.     I    ->   AS    :   I, C, N3
5.    AS    ->    I    :   AS, {AS, S, N3, pk(C)}sk(AS)
6.     S    ->    C    :   S, C, {S, inc(N2)}pk(C)
```

**3.** Lowe outlined (see [CJ97]) that a malicious `C` can replay the message `3` (the first message concerning `S`) several times, with new values of `T` and `L`, to restart authentication with an old value of `N2`.

### See also

Hwang and Chen modified SPLICE/AS, Clark and Jacob modified Hwang and Chen modified SPLICE/AS.

## Hwang and Chen modified SPLICE/AS

**Author(s):** Tzonelih Hwang and Yung-Hsiang Chen 1995
*Last modified November 11, 2002*

**Summary:** This modified version correct two flaws in SPLICE/AS. Mutual authentication protocol with public key cryptography with a certification authority signing and distributing public keys.

### Protocol specification (in common syntax)

```
S, C, AS :    principal
N1, N2, N3 :  nonce
T :           timestamp
L :           lifetime
pk, sk :      principal -> key (keypair)

1.    C   ->  AS  :   C, S, N1
2.    AS  ->  C   :   AS, {AS, C, N1, S, pk(S)}sk(AS)
3.    C   ->  S   :   C, S, {C, T, L, {N2}pk(S)}sk(C)
4.    S   ->  AS  :   S, C, N3
5.    AS  ->  S   :   AS, {AS, S, N3, C, pk(C)}sk(AS)
6.    S   ->  C   :   S, C, {S, inc(N2)}pk(C)
```

### Description of the protocol rules

See SPLICE/AS. Note that the name of the owner of the public key is included in certificate to overcomes the flaws of SPLICE/AS presented in [HC95] (i.e. a certificate for the public key `pk(S)` is here `{AS, C, N1, S, pk(S)}sk(AS)` rather than `{AS, C, N1, pk(S)}sk(AS)` in SPLICE/AS).

### Requirements

See SPLICE/AS.

### References

[HC95].

### Claimed attacks

[CJ95]. Only the messages `3` and `6` are relevant in this attack, in which the intruder `I` learn the secret `N2`. This attack concerns both the secrecy of `N2` and its authenticity.

```
 1.      C     ->    AS    :    C, S, N1
 2.     AS     ->    C     :    AS, {AS, C, N1, S, pk(S)}sk(AS)
 3.      C     ->   I(S)   :    C, S, {C, T, L, {N2}pk(S)}sk(C)
 3.      I     ->    S     :    I, S, {I, T, L, {N2}pk(S)}sk(I)
 4.      S     ->    AS    :    S, I, N3
 5.     AS     ->    S     :    AS, {AS, S, N3, I, pk(I)}sk(AS)
 6.      S     ->    I     :    S, I, {S, inc(N2)}pk(I)
 1.      I     ->    AS    :    I, C, N1'
 2.     AS     ->    I     :    AS, {AS, I, N1', C, pk(C)}sk(AS)
 6.    I(S)    ->    C     :    S, C, {S, inc(N2)}pk(C)
```

### See also

SPLICE/AS, Clark and Jacob modified Hwang and Chen modified SPLICE/AS.

## Clark and Jacob modified Hwang and Chen modified SPLICE/AS

**Author(s):** 1995
*Last modified November 11, 2002*

**Summary:** This modified version corrects a flaws in Hwang and Chen modified SPLICE/AS. Mutual authentication protocol with public key cryptography with a certification authority signing and distributing public keys.

## Protocol specification (in common syntax)

```
S, C, AS :    principal
N1, N2, N3 :  nonce
T :           timestamp
L :           lifetime
pk, sk :      principal -> key (keypair)

1.    C   ->  AS   :    C, S, N1
2.    AS  ->  C    :    AS, {AS, C, N1, S, pk(S)}sk(AS)
3.    C   ->  S    :    C, S, {T, L, {C, N2}pk(S)}sk(C)
4.    S   ->  AS   :    S, C, N3
5.    AS  ->  S    :    AS, {AS, S, N3, C, pk(C)}sk(AS)
6.    S   ->  C    :    S, C, {inc(N2)}pk(C)
```

## Remark

This protocol is an optimised version of the following modification of Hwang and Chen modified SPLICE/AS:

```
1.    C   ->  AS   :    C, S, N1
2.    AS  ->  C    :    AS, {AS, C, N1, S, pk(S)}sk(AS)
3.    C   ->  S    :    C, S, {C, T, L, {C, N2}pk(S)}sk(C)
4.    S   ->  AS   :    S, C, N3
5.    AS  ->  S    :    AS, {AS, S, N3, C, pk(C)}sk(AS)
6.    S   ->  C    :    S, C, {S, inc(N2)}pk(C)
```

The messages 3 and 6 are optimised by suppressing some redundancies: the redundant C is not included in the signed part of message 3 and S in not included in the cipher of message 6

## Description of the protocol rules

See SPLICE/AS. The difference with Hwang and Chen modified SPLICE/AS is in messages Note that the name of the owner of the public key is included in certificate to overcomes the flaws of SPLICE/AS presented in [HC95] (i.e. a certificate for the public key pk(S) is here {AS, C, N1, S, pk(S)}sk(AS) rather than {AS, C, N1, pk(S)}sk(AS) in SPLICE/AS).

## Requirements

See SPLICE/AS.

### References

[CJ95].

### Claimed attacks

Lowe [Low97] demonstrate a multiplicity attack on this protocol, where I impersonates C in a new session ii, by replaying message 3 of session i. I does however not learn N2.

```
 i.1.      C     ->   AS    :   C, S, N1
 i.2.      AS    ->   C     :   AS, {AS, C, N1, S, pk(S)}sk(AS)
 i.3.      C     ->   S     :   C, S, {T, L, {C, N2}pk(S)}sk(C)
 i.4.      S     ->   AS    :   S, C, N3
 i.5.      AS    ->   S     :   AS, {AS, S, N3, C, pk(C)}sk(AS)
 i.6.      S     ->   C     :   S, C, {inc(N2)}pk(C)
 ii.3.     I(C)  ->   S     :   C, S, {T, L, {C, N2}pk(S)}sk(C)
 ii.4.     S     ->   AS    :   S, C, N'3
 ii.5.     AS    ->   S     :   AS, {AS, S, N'3, C, pk(C)}sk(AS)
 ii.6.     S     ->   I(C)  :   S, C, {inc(N2)}pk(C)
```

Lowe suggests in [Low97] to add a nonce challenge to prevent this attack.

### See also

SPLICE/AS, Hwang and Chen modified SPLICE/AS.

# TMN

**Author(s):** M. Tatebayashi, N. Matsuzaki, and D.B. Newman 1989
*Last modified January 16, 2003*

**Summary:** Distribution of a fresh symmetric key and authentication. Symmetric keys, trusted server and public keys (only the public key of the server is used).

### Protocol specification (in common syntax)

```
 A, B, S :  principal
 Ka, Kb :   key
 PK, SK :   principal -> key (keypair)
 1.    A  ->  S  :   B, {Ka}PK(S)
 2.    S  ->  B  :   A
 3.    B  ->  S  :   A, {Kb}PK(S)
 4.    S  ->  A  :   B, {Kb}Ka
```

### Description of the protocol rules

We assume that both A and B initially know the public key PK(S) of S.

Ka, Kb are session symmetric keys freshly created by A, resp. B.

In message 4, Kb is encrypted using a symmetric key algorithm with the key Ka. Hence, the encryption operators used in 4 on one hand and in 1 and 3 on the other hand differ (though the notation is the same).

### Remark

The binary operator {Kb}Ka in the last message can be intepreted either by a xor operator or by another symmetric key encryption algorithm, according to the implementation of the protocol.

This choice may be important, as the attack 4. below shows.

### Requirements

The protocol must guaranty the secrecy of the new shared key Kb: in every session, the value of Kb must be known only by the participants playing the roles of A and B in that session.

The protocol must guaranty the secrecy of the auxiliary fresh key Ka: in every session, the value of Ka must be known only by the participants playing the roles of A and S in that session.

### References

[TMN89], see also [LR97].

### Claimed attacks

**1.** [LR97]. Authentication and secrecy failure: the intruder I impersonates A, and uses a session auxiliary key Ki of his choice to learn the established session key Kb in the last message.

```
1.   I(A)  ->    S     :   B, {Ki}PK(S)
2.    S    ->    B     :   A
3.    B    ->    S     :   A, {Kb}PK(S)
4.    S    ->   I(A)   :   B, {Kb}Ki
```

Note that this is a very simple attack without parallel session or replay.

**2.** [LR97]. Authentication failure: the intruder `I` impersonates `B` and establishes a new session key `Ki` of his choice.

```
1.    A    ->    S     :   B, {Ka}PK(S)
2.    S    ->   I(B)   :   A
3.   I(B)  ->    S     :   A, {Ki}PK(S)
4.    S    ->   I(A)   :   B, {Ki}Ka
```

This attack demonstrates actually more than an authentication flaw, because the established session key is known to the intruder. With the following additional fifth message representing further communications between `A` and `B` using the new established shared key `Kb`:

```
5.    A   ->  B   :    {X}Kb
```

the protocol would not guaranty the secrecy of `X` as expected.

**3.** [LR97]. Parallel session and replay attack combining the above attacks 1 and 2. Secrecy and authentication failure: at the end of the second session, the intruder knows the established session key `Kb`.

```
i.1.     I(A)   ->    S     :    B, {Ki}PK(S)
i.2.      S     ->    B     :    A
i.3.      B     ->    S     :    A, {Kb}PK(S)
i.4.      S     ->   I(A)   :    B, {Kb}Ki
ii.1.     A     ->    S     :    B, {Ka}PK(S)
ii.2.     S     ->   I(B)   :    A
ii.3.    I(B)   ->    S     :    A, {Kb}PK(S)
ii.4.     S     ->   I(A)   :    B, {Kb}Ka
```

Note that after this attack, `A` and `B` shall communicate with the compromised session key `Kb`. This was not the case with attacks 1 and 2, because during these attacks, the authentication had been performed only with one honest principal.

**4.** The following secrecy attack, described in [Sim88, Sim94], see also [TMN89], doesn't rely on an authentication failure but on algebraic properties of the encryption method.

It assumes that the symmetric key encryption is performed by a operator `+` such that:

```
(x+y)+y  =  x     (1)
x+(x+y)  =  y     (1')
```

Hence, the protocol reads:

```
1.   A  ->  S  :  B, {Ka}PK(S)
2.   S  ->  B  :  A
3.   B  ->  S  :  A, {Kb}PK(S)
4.   S  ->  A  :  B, Kb + Ka
```

We A, knowing Ka, receives the message 4, he can obtain Kb by (1).

Let * be a multiplication operator such that the public key encryption algorithm verifies, for all public key PK(U):

```
{x * {y}PK(U)}PK(U) = {x*y}PK(U)    (2)
```

Moreover, we assume a partial division operator (associated to *).

These hypotheses are satisfied e.g. if the following choices are made for the operators:

- + is xor,
- {x}n is x^3 mod n (with x < n),
- * is integer multiplication.

The attack is then the following. The intruder I has learned the message 3 from a first session i, and will use the server S as an oracle in a second session ii to learn the key Kb. D is the identity of an honest principal (which can be A or B or anyone else).

```
i.3.     B    ->  I(S)  :  A, {Kb}PK(S)
ii.1.    I    ->   S    :  D, {Ki * {Kb}PK(S)}PK(S) ( = {Ki*Kb}PK(S) by (2) )
ii.2.    S    ->  I(D)  :  I
ii.3.   I(D)  ->   S    :  I, {Kd}PK(S)
ii.4.    S    ->   I    :  D, Kd + (Ki * Kb)
```

Ki and Kd are arbitrary values chosen by I.

After receiving ii.4, I can compute Ki * Kb = Kd + (Kd + (Ki * Kb)), using (1'), and hence Kb.

Note that in this attack, the server S cannot detect the replay of {Kb}PK(S) in message ii.1 because it is multiplied with the arbitrary value Ki.

## Comment sent by Ralf Treinen (*13/01/2003*)

Ralf Treinen has submitted the above claimed attack number 4.

## Wired Equivalent Privacy Protocol

**Author(s):**

**Summary:** The Wired Equivalent Privacy (WEP) protocol, described in [80299], is used to protect data during wireless transmission.

### Protocol specification (in common syntax)

```
A, B :  principal
Kab :   symkey
RC4 :   message, symkey -> message
C :     message -> message

1.    A  ->  B  :     v, ((M,C(M)) xor RC4(v,Kab))
```

### Description of the protocol rules

To encrypt the message M, A applies the operator xor to RC4(v,Kab) and (M,C(M)) where C(M) is the *integrity checksum* of the message M and RC4 is a function modeling the RC4 algorithm which is used to generate a *keystream* (*i.e.* a long sequence of pseudo-random bytes) from the initial vector v and the secret key Kab shared between A and B. To decrypt the received message, B computes RC4(v,Kab) and after applying exclusive or, he obtains (M,C(M)) and can verify that the checksum is correct.

The properties of exclusive or are:

$$\begin{aligned}
\texttt{x xor (y xor z)} &= \texttt{(x xor y) xor z} \quad \text{(E1)}\\
\texttt{x xor y} &= \texttt{y xor x} \quad \text{(E2)}\\
\texttt{x xor 0} &= \texttt{x} \quad \text{(E3)}\\
\texttt{x xor x} &= \texttt{0} \quad \text{(E4)}
\end{aligned}$$

### References

[80299]

### Claimed attacks

We present below attacks given in [BGW01] that require the following properties:

```
         C(x xor y)  =  C(x) xor C(Y)  (E5)
(x1,y1) xor (x2,y2)  =  (x1 xor x2,y1 xor y2)  (E6)
```

According to [BGW01], (E5) is a general property of CRC checksum.

The first attack uses the fact that encrypting two messages P1 and P2 with the same initial vector v and with the same key k can reveal information. Indeed, we have the following equalities between the ciphers C1 and C2 and their associated plain text P1 and P2:

```
C1 xor C2  =  ((P1,C(P1)) xor RC4(v,k)) xor ((P2,C(P2)) xor RC4(v,k))
           =  ((P1,C(P1)) xor (P2,C(P2))) xor (RC4(v,k)xor RC4(v,k))     (E1)(E2)
           =  (P1,C(P1)) xor (P2,C(P2))                                  (E3)(E4)
```

This allows an intruder who knows a plain text P1 and its cipher C1 to decrypt any cipher C2. Indeed, thanks to this equality, the intruder can easily get (P2,C(P2)) and obtain the plaintext P2.

The second attack allows the intruder controlled modifications to a cipher text without disrupting the checksum. Assume that the intruder has intercepted (M,C(M))xor RC4(v,Kab) and knows D. He can now obtain the cipher text associated to the message Mxor D by computin g:

```
((M,C(M)) xor RC4(v,Kab)) xor (D,C(D))  =  RC4(v,Kab)xor((M,C(M)) xor (D,C(D)))     (
                                        =  RC4(v,Kab) xor (M xor D,C(M) xor C(D))    (
                                        =  RC4(v,Kab) xor (M xor D,C(M xor D))       (
```

Notice that this attack can be applied without full knowledge of M. For example, to flip the first bit of M, the attacker can set D = 100...0. Now, if the intruder knows the plaintext M (and its associated cipher) he can generate the ciphertext associated to any message he wants.

## Wide Mouthed Frog

**Author(s):** Michael Burrows 1989
*Last modified November 20, 2002*

**Summary:** Distribution of a fresh shared key. Symmetric key cryptography with server and timestamps.

## Protocol specification (in common syntax)

```
A, S :          principal
Kas, Kbs, Kab : symkey
Ta, Ts :        timestamp

1.   A  ->  S  :   A, {Ta, B, Kab}Kas
2.   S  ->  B  :   {Ts, A, Kab}Kbs
```

## Description of the protocol rules

Some explanations quoted from [BAN89]:

> "It is assumed that the encryption is done in such a way that we know the whole message was sent at once. If two separate encrypted sections are included in one message, we treat them as though they arrived in separate messages. A message cannot be understood by a principal who does not know the key (or, in the case of public-key cryptography, by a principal who does not know the inverse of the key); the key cannot be deduced from the encrypted message. Each encrypted message contains sufficient redundancy to allow a principal who decrypts it to verify that he has used the right key. In addition, messages contain sufficient information for a principal to detect (and ignore) his own messages."

> "A sends a session key to S, including a timestamp Ta. S checks that the first message is timely, and if it is, it forwards the message to B, together with its own timestamp Ts. B then checks that the timestamp from S is later than any other it has received from S."

## Requirements

The protocol must guaranty the secrecy of the new shared key Kab: in every session, the value of Kab must be known only by the participants playing the roles of A and B and S.

The protocol must guaranty the authenticity of Kab: in every session, on reception of message 2, B must be ensured that the key Kab in the message has been created by S in the same session on behalf of A.

## References

[BAN89]

## Claimed proofs

[BAN89]

## Claimed attacks

**1.** [AN95]. By replaying the second message within an appropriate time window, the intruder `I` can make the server `S` update the timestamp of an non-fresh key `Kab`. This way, he can extend the life time of a (possibly compromised) key `Kab` as wanted, whereas `A` and `B` think that it has expired and has been destroyed.

```
i.1.       A    ->  S  :   A, {Ta, B, Kab}Kas
i.2.       S    ->  B  :   {Ts, A, Kab}Kbs
ii.1.     I(B)  ->  S  :   B, {Ts, A, Kab}Kbs
ii.2.      S    ->  A  :   {T's, B, Kab}Kas
iii.1.    I(A)  ->  S  :   A, {T's, B, Kab}Kas
iii.2.     S    ->  B  :   {T''s, A, Kab}Kbs
....
```

**2.** [Low97]. In this attack, `B` thinks that `A` has established two sessions with him, when `A` thinks he has established only one session.

```
i.1.     A  ->  S  :   A, {Ta, B, Kab}Kas
i.2.     S  ->  B  :   {Ts, A, Kab}Kbs
ii.2.    S  ->  B  :   {Ts, A, Kab}Kbs
```

[Low97] proposes a correction of the protocol which is described in Lowe modified Wide Mouthed Frog.

## Comment sent by Martin Abadi (*November 18, 2002*)

The [AN95] and [Low97] "attacks" fail, because of the protocol features described in the quotations above. The "attacks" may work only against (deliberately or unintentionally) weakened variants of the protocol.

## See also

Lowe modified Wide Mouthed Frog

## Lowe modified Wide Mouthed Frog

**Author(s):** Gavin Lowe 1997
*Last modified November 20, 2002*

**Summary:** An modified version of Wide Mouthed Frog. Exchanged of a fresh shared key. Symmetric key cryptography with server and timestamps.

### Protocol specification (in common syntax)

```
A, S :            principal
Kas, Kbs, Kab :   symkey
Nb :              nonce
Ta, Ts :          timestamp
suc :             nonce -> nonce

1.   A  ->  S  :   A, {Ta, B, Kab}Kas
2.   S  ->  B  :   {Ts, A, Kab}Kbs
3.   B  ->  A  :   {Nb}Kab
4.   A  ->  B  :   {succ(Nb)}Kab
```

### Description of the protocol rules

Two messages have been appened to Wide Mouthed Frog for mutual authentification of `A` and `B` (*nonce handshake*).

### Remark

The two final messages were added by Lowe to the Wide Mouthed Frog protocol to prevent an attack claimed in [Low97] which actually fails against the complete original specification of the protocol in [BAN89], see Wide Mouthed Frog.

### Requirements

See Wide Mouthed Frog.

### References

[Low97]

### See also

Wide Mouthed Frog

## Woo and Lam Mutual Authentication

**Author(s):** T.Y.C Woo and S.S. Lam 1994
*Last modified November 10, 2002*

**Summary:** Key distribution and mutual authentication with trusted server and symmetric keys.

### Protocol specification (in common syntax)

```
P, Q, S :         principal
Kps, Kqs, Kpq :   key
N1, N2 :          number

1.    P  ->  Q  :    P, N1
2.    Q  ->  P  :    Q, N2
3.    P  ->  Q  :    {P, Q, N1, N2}Kps
4.    Q  ->  S  :    {P, Q, N1, N2}Kps, {P, Q, N1, N2}Kqs
5.    S  ->  Q  :    {Q, N1, N2, Kpq}Kps, {P, N1, N2,Kpq}Kqs
6.    Q  ->  P  :    {Q, N1, N2, Kpq}Kps, {N1, N2}Kpq
7.    P  ->  Q  :    {N2}Kpq
```

### Description of the protocol rules

Kpq is a fresh symmetric key created at message 5 by the server S.

N1 and N2 are nonces.

Kps and Kqs are symmetric keys whose values are initially known only by P and S, respectively P and S.

### Requirements

The protocol must guaranty the secrecy of Kpq: in every session, the value of Kpq must be known only by the participants playing the roles of P, Q and S.

The protocol must also ensures mutual authentication of P and Q.

### References

[WL94]

### Claimed attacks

**1.** Parallel session replay attack, [CJ], and [CJ97]. In this attack, the intruder `I` initiates a session `ii` in order to make `P` accept a non-fresh key.

```
 i.1.      P    ->   I    :   P, N1
 ii.1.     I    ->   P    :   I, N1
 ii.2.     P    ->   I    :   P, N2
 i.2.      I    ->   P    :   I, N2
 i.3.      P    ->   I    :   {P, I, N1, N2}Kps
 i.4.      I    ->   S    :   {P, I, N1, N2}Kps, {P, I, N1, N2}Kis
 i.5.      S    ->   I    :   {I, N1, N2, Kpi}Kps, {P, N1, N2, Kpi}Kis
 i.6.      I    ->   P    :   {I, N1, N2, Kpi}Kps, {N1, N2}Kpi
 i.7.      P    ->   I    :   {N2}Kpi
 ii.3.     I    ->   P    :   {I, P, N1, N2}Kis
 ii.4.     P    ->  I(S)  :   {I, P, N1, N2}Kis, {I, P, N1, N2}Kps
 ii.5.    I(S)  ->   P    :   {I, N1, N2, Kpi}Kis, {I, N1, N2, Kpi}Kps
 ii.6.     P    ->   I    :   {P, N1, N2, Kpi}Kis, {N1, N2}Kpi
 ii.7.     I    ->   P    :   {N2}Kpi
```

**2.** [Low96]. `bit-string` represent an arbitrary number.

```
 i.1.     I(P)  ->   Q    :   P, Q
 i.2.      Q    ->  I(P)  :   Q, N2
 i.3.     I(P)  ->   Q    :   bit-string
 i.4.      Q    ->  I(S)  :   bit-string, {P, Q, Q, N2}Kps
 ii.1.    I(P)  ->   Q    :   P, N2
 ii.2.     Q    ->  I(P)  :   Q, N3
 ii.3.    I(P)  ->   Q    :   bit-string'
 ii.4.     Q    ->  I(S)  :   bit-string', {P, Q, N2, N3}Kps
 i.5.     I(S)  ->   Q    :   bit-string'', {P, Q, N2, N3}Kps
 i.6.      Q    ->  I(P)  :   bit-string'', {Q, N2}N3
 i.7.     I(P)  ->   Q    :   {N2}N3
```

# Woo and Lam Pi

**Author(s):** Woo, Lam 1994
*Last modified October 27, 2001*

**Summary:** One way authentification protocol with public keys and trusted server, simplification of Woo and Lam Pi 3, Woo and Lam Pi 2, Woo and Lam Pi 1, and Woo and Lam Pi f.

### Protocol specification (in common syntax)

```
A, B, S :   principal
Nb :        nonce
Kas, Kbs :  skey

1..    A  ->  B   :    A
2..    B  ->  A   :    Nb
3..    A  ->  B   :    {Nb}Kas
4..    B  ->  S   :    {A, {Nb}Kas}Kbs
5..    S  ->  B   :    {Nb}Kbs
```

### Requirements

see Woo and Lam Pi f.

### References

[WL94], [CJ97].

### Claimed attacks

### See also

Woo and Lam Pi f, Woo and Lam Pi 1, Woo and Lam Pi 2, Woo and Lam Pi 3.

## Woo and Lam Pi 1

**Author(s):** Woo, Lam 1994
*Last modified October 27, 2001*

**Summary:** One way authentification protocol with public keys and trusted server, simplification of Woo and Lam Pi f.

### Protocol specification (in common syntax)

```
A, B, S :   principal
Nb :        nonce
Kas, Kbs :  skey

1..    A  ->  B   :    A
2..    B  ->  A   :    Nb
3..    A  ->  B   :    {A,B,Nb}Kas
4..    B  ->  S   :    {A, B, {A, B, Nb}Kas}Kbs
5..    S  ->  B   :    {A, B, Nb}Kbs
```

### Requirements

see Woo and Lam Pi f.

### References

[WL94], [CJ97].

### See also

Woo and Lam Pi f, Woo and Lam Pi 2, Woo and Lam Pi 3, Woo and Lam Pi.

## Woo and Lam Pi 2

**Author(s):** Woo, Lam 1994
*Last modified October 27, 2001*

**Summary:** One way authentification protocol with public keys and trusted server, simplification of Woo and Lam Pi 1 and Woo and Lam Pi f.

### Protocol specification (in common syntax)

```
A, B, S :   principal
Nb :        nonce
Kas, Kbs :  skey
```

```
1..   A  ->  B  :   A
2..   B  ->  A  :   Nb
3..   A  ->  B  :   {A,Nb}Kas
4..   B  ->  S  :   {A, {A, Nb}Kas}Kbs
5..   S  ->  B  :   {A, Nb}Kbs
```

### Requirements

see Woo and Lam Pi f.

### References

[WL94], [CJ97].

### See also

Woo and Lam Pi f, Woo and Lam Pi 1, Woo and Lam Pi 3, Woo and Lam Pi.

## Woo and Lam Pi 3

**Author(s):** Woo, Lam 1994
*Last modified October 27, 2001*

**Summary:** One way authentification protocol with public keys and trusted server, simplification of Woo and Lam Pi 2, Woo and Lam Pi 1, and Woo and Lam Pi f.

### Protocol specification (in common syntax)

```
A, B, S :   principal
Nb :        nonce
Kas, Kbs :  skey

1..   A  ->  B  :   A
2..   B  ->  A  :   Nb
3..   A  ->  B  :   {Nb}Kas
4..   B  ->  S  :   {A, {Nb}Kas}Kbs
5..   S  ->  B  :   {A, Nb}Kbs
```

### Requirements

see Woo and Lam Pi f.

### References

[WL94], [CJ97].

### See also

Woo and Lam Pi f, Woo and Lam Pi 1, Woo and Lam Pi 2, Woo and Lam Pi.

## Woo and Lam Pi f

**Author(s):** Woo, Lam 1994
*Last modified October 27, 2001*

**Summary:** One way authentification protocol with public keys and trusted server.

### Protocol specification (in common syntax)

```
A, B, S :  principal
shared :   (principal, principal):key
Nb :       nonce

1..    A  ->  B   :   A
2..    B  ->  A   :   Nb
3..    A  ->  B   :   {A,B,Nb}shared(A, S)
4..    B  ->  S   :   {A, B, Nb, {A, B, Nb}shared(A, S)}shared(B, S)
5..    S  ->  B   :   {A, B, Nb}shared(B, S)
```

### Description of the protocol rules

`shared(A, S)` is a long term symmetric key shared by `A` and `S`. Initially, `A` only knowns `shared(A, S)` and the name of `B`, `B` only knowns `shared(B, S)` and `S` knowns all shared keys, i.e. `S` given any principal's name `X`, `S` knowns `shared(X, S)`, or in other terms, `S` knows the "function" `shared`.

### Requirements

Woo and Lam give in [WL94] the following definition of correctness for this protocol:

whenever the principal `B` finishes the execution of the protocol, the initiator of the protocol execution is in fact the principal `A` claimed in message `1`.

### References

[WL94], [CJ97].

### Claimed proofs

[WL94]

### Claimed attacks

No known attacks.

### See also

Woo and Lam Pi 1, Woo and Lam Pi 2, Woo and Lam Pi 3, Woo and Lam Pi.

## Yahalom

**Author(s):** Yahalom 0, 1988
*Last modified October 4, 2002*

**Summary:** Distribution of a fresh symmetric shared key by a trusted server and mutual authentication. Symmetric keys and trusted server.

### Protocol specification (in common syntax)

```
A, B, S :        principal
Na, Nb :         number fresh
Kas, Kbs, Kab :  key
```

```
A knows :  A, B, S, Kas
B knows :  B, S, Kbs
S knows :  S, A, B, Kas, Kbs

1.    A  ->  B   :     A, Na
2.    B  ->  S   :     B, {A, Na, Nb}Kbs
3.    S  ->  A   :     {B, Kab, Na, Nb}Kas, {A, Kab}Kbs
4.    A  ->  B   :     {A, Kab}Kbs, {Nb}Kab
```

### Description of the protocol rules

The fresh symmetric shared key `Kab` is created by the server `S` and sent encrypted, in message `3` both to `A` (directly) and to `B` (indirectly).

### Requirements

The protocol must guaranty the secrecy of `Kab`: in every session, the value of `Kab` must be known only by the participants playing the roles of `A`, `B` and `S`.

`A` must be also properly authentified to `B`.

### References

This version of the Yahalom protocol is the one found in [BAN89] (cited as personal communication in this paper).

It is also presented in [CJ97].

### Claimed proofs

[BAN89], [Pau01]

### See also

BAN simplified version of Yahalom,
Paulson's strengthened version of Yahalom.

## BAN simplified version of Yahalom

**Author(s):** Burrows Abadi Needham 0, 1989
*Last modified October 9, 2002*

**Summary:** An amended version of the Yahalom protocol, presented in the BAN logic paper. Symmetric keys and trusted server.

## Protocol specification (in common syntax)

```
A, B, S :        principal
Na, Nb :         number fresh
Kas, Kbs, Kab :  key

A knows :  A, B, S, Kas
B knows :  B, S, Kbs
S knows :  S, A, B, Kas, Kbs

1.   A  ->  B   :    A, Na
2.   B  ->  S   :    B, Nb, {A, Na}Kbs
3.   S  ->  A   :    Nb, {B, Kab, Na}Kas, {A, Kab, Nb}Kbs
4.   A  ->  B   :    {A, Kab, Nb}Kbs, {Nb}Kab
```

## Description of the protocol rules

Compared to the original version of the Yahalom protocol, the nonce `Nb` is added to the second cipher of message `3`, to prevent a malicious `A` to reuse an old value of `Kab`.

Also, `Nb` is sent in cleartext in message `2`, which makes possible the attacks below.

## Requirements

See Yahalom.

## References

This simplified version of the Yahalom protocol was proposed in [BAN89].

## Claimed proofs

[BAN89]

## Claimed attacks

Replay attack with interleaving and type error in [Syv94].

```
i.1.       A    ->  I(B)  :   A, Na
i.2.       B    ->  I(S)  :   B, Nb, {A, Na}Kbs
ii.1.    I(A)   ->   B    :   A, Na, Nb
ii.2.      B    ->  I(S)  :   B, Nb, {A, Na, Nb}Kbs
i.3.                          Omitted
i.4.     I(A)   ->   B    :   {A, Na, Nb}Kbs, {Nb}Na
```

In the message 1 of session `ii`, the pair `Na, Nb` is used as a nonce `N'a`, and in the last message of session `i`, `Na` is used as the key `Kab`.

A second replay attack is described in the same paper [Syv94].

```
i.1.        A    ->  I(B)  :   A, Na
ii.1.     I(B)   ->   A    :   B, Na
ii.2.       A    ->  I(S)  :   A, N'a, {B, Na}Kas
iii.1.                         Omitted
iii.2.    I(A)   ->   S    :   A, Na, {B, Na}Kas
iii.3.      S    ->  I(B)  :   Na, {A, Kab, Na}Kbs, {B, Kab, Na}Kas
i.2.                           Omitted
i.3.      I(S)   ->   A    :   Ni, {B, Kab, Na}Kas, {A, Kab, Na}Kbs
i.4.        A    ->  I(B)  :   {A, Kab, Na}Kbs, {Ni}Kab
```

### See also

Yahalom,
Paulson's strengthened version of Yahalom.

## Lowe's modified version of Yahalom

**Author(s):** Paulson 0, 2001
*Last modified October 4, 2002*

**Summary:** Lowe's modified version of the Yahalom protocol. Symmetric keys and trusted server.

### Protocol specification (in common syntax)

```
A, B, S :          principal
Na, Nb :           number fresh
Kas, Kbs, Kab :    key

A knows :  A, B, S, Kas
B knows :  B, S, Kbs
S knows :  S, A, B, Kas, Kbs
```

```
1.    A  ->  B  :    A, Na
2.    B  ->  S  :    {A, Na, Nb}Kbs
3.    S  ->  A  :    {B, Kab, Na, Nb}Kas
4.    S  ->  B  :    {A, Kab}Kbs
5.    A  ->  B  :    {A, B, S, Nb}Kab
```

### Remark

This version of the Yahalom protocol is presented in [Low98] to illustrate a verification technique by model checking.

### Requirements

See Yahalom.

### References

[Low98]

### Claimed proofs

[Low98]

### See also

Yahalom,
BAN simplified version of Yahalom,
Paulson's strengthened version of Yahalom.

## Paulson's strengthened version of Yahalom

**Author(s):** Paulson 0, 2001
*Last modified October 4, 2002*

**Summary:** Paulson's modified version of the Yahalom protocol. Symmetric keys and trusted server.

## Protocol specification (in common syntax)

```
A, B, S :        principal
Na, Nb :         number fresh
Kas, Kbs, Kab :  key

A knows :  A, B, S, Kas
B knows :  B, S, Kbs
S knows :  S, A, B, Kas, Kbs

1.    A  ->  B   :    A, Na
2.    B  ->  S   :    B, Nb, {A, Na}Kbs
3.    S  ->  A   :    Nb, {B, Kab, Na}Kas, {A, B, Kab, Nb}Kbs
4.    A  ->  B   :    {A, B, Kab, Nb}Kbs, {Nb}Kab
```

## Description of the protocol rules

To prevent the attacks [Syv94] of to BAN simplified version of Yahalom protocol, the name of `B` has been added to the cipher sent by `S` in message `3` and transmitted by `A` in message `4`.

## Requirements

See Yahalom.

## References

[Pau01]

## Claimed proofs

[Pau01]

## See also

Yahalom,
BAN simplified version of Yahalom

## Citations

[80299] IEEE 802.11 Local and Metropolitan Area Networks: Wireless LAN Medium Acess Control (MAC) and Physical (PHY) Specifications, 1999.

[AN95] R. Anderson and R. Needham. Programming satan's computer, 1995.

[AN96] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[BAN89] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. Technical Report 39, Digital Systems Research Center, february 1989.

[Bel01] Giampaolo Bella. Mechanising a protocol for smart cards. In *Proc. of e-Smart 2001, international conference on research in smart cards*, LNCS. Springer-Verlag, september 2001.

[BGW01] N. Borisov, I. Goldberg, and D. Wagner. Intercepting mobile communications: The insecurity of 802.11. In *Proc. 7th Annual International Conference on Mobile Computing and Networking (MOBICOM'01)*, pages 180–188, Rome (Italy), 2001. ACM Press.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In IEEE, editor, *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, june 2001.

[BMS02] Scott Bradner, Allison Mankin, and Jeffrey I. Schiller. A framework for purpose built keys (PBK). Internet draft, November 2002.

[BO97] J. Bull and D. J. Otway. The authentication protocol. Technical Report DRA/CIS3/PROJ/CORBA/SC/1/CSM/436-04/03, Defence Research Agency, 1997.

[BR95] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution– the three party case. In *Proceedings 27th Annual Symposium on the Theory of Computing*, ACM, pages 57–66, 1995.

[CCI87] CCITT. The directory authentification framework. Draft Recommendation X.509, 1987. Version 7.

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Proc. 14th Annual International Cryptology Conference (CRYPTO'94)*, volume 963 of *LNCS*, pages 174–187, Santa Barbara (California, USA), 1994. Springer-Verlag.

[CJ] John Clark and Jeremy Jacob. Freshness is not enough : Note on trusted nonce generation and malicious principals. attack on a mutual authentification protocol by Woo and Lam.

[CJ95] John A Clark and Jeremy L Jacob. On the security of recent protocols. *Information processing Letters*, 56:151–155, 1995.

[CJ97] John Clark and Jeremy Jacob. A survey of authentication protocol literature, November 1997.

[CKS01] R. Chadha, M.I. Kanovich, and A. Scedrov. Inductive methods and contract-signing protocols. In P. Samarati, editor, *8-th ACM Conference on Computer and Communications Security*, pages 176–185. ACM Press, November 2001.

[DH76] W. Diffie and M. Helman. New directions in cryptography. *IEEE Transactions on Information Society*, 22(6):644–654, november 1976.

[DS81] D. Denning and G. Sacco. Timestamps in key distributed protocols. *Communication of the ACM*, 24(8):533–535, 1981.

[GJM99] J. A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Advances in Cryptology: Proceedings of Crypto'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 449–466. Springer-Verlag, 1999.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proc. 17th annual ACM Symposium on Theory of Computing*, pages 291–304. ACM Press, 1985.

[Gol01] O. Goldreich. *Foundations of Cryptography.* Cambridge University Press, 2001.

[Gon89] Li Gong. Using one-way functions for authentication. *Computer Communication Review*, 19(5):8–11, october 1989.

[GTTT03] Thomas Genet, Yan-Mei Tang-Talpin, and Valérie Viet Triem Tong. Verification of copy-protection cryptographic protocol using approximations of term rewriting systems. In *Proc. of WITS'03, Workshop on Issues in the Theory of Security*, 2003.

[HC95] Tzonelih Hwang and Yung-Hsiang Chen. On the security of splice/as : The authentication system in wide internet. *Information Processing Letters*, 53:97–101, 1995.

[HLL$^+$95] Tzonelih Hwang, Narn-Yoh Lee, Chuang-Ming Li, Ming-Yung Ko, and Yung-Hsiang Chen. Two attacks on neumann-stubblebine authentication protocols. *Information Processing Letters*, 53:103 – 107, 1995.

[Hui98] Christian Huitema. *IPv6 The New Internet Protocol.* Prentice Hall PTR, 1998.

[JHC$^+$98] Rob Jerdonek, Peter Honeyman, Kevin Coffman, Kim Rees, and Kip Wheeler. Implementation of a provably secure, smartcard-based key distribution protocol. In *In Proceedings of the Third Smart Card Research and Advanced Application Conference*, 1998.

[KC95] I Lung Kao and Randy Chow. An efficient and secure authentication protocol using uncertified keys. *Operating Systems Review*, 29(3):14–21, 1995.

[KR02] Steve Kremer and Jean-François Raskin. Game analysis of abuse-free contract signing. In Steve Schneider, editor, *15th Computer Security Foundations Workshop*, pages 206–220, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Computer Society.

[KSL92] Axel Kehne, Jürgen Schönwälder, and Horst Langendörfer. Multiple authentications with a nonce-based protocol using generalized timestamps. In *Proc. ICCC '92*, Genua, 1992.

[lM90] Colin l'Anson and Chris Mitchell. Security defects in the ccitt recomendation x.509 - the directory authentication framework. *Computer Communication Review*, 20(2):30–34, april 1990.

[Low95] Gavin Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–136, November 1995.

[Low96] Gavin Lowe. Some new attacks upon security protocols. In IEEE Computer Society Press, editor, *In Proceedings of the Computer Security Foundations Workshop VIII*, 1996.

[Low97] Gavin Lowe. A family of attacks upon authentication protocols. Technical Report 1997/5, Department of Mathematics and Computer Science, University of Leicester, 1997.

[Low98] Gavin Lowe. Towards a completeness result for model checking of security protocols. Technical Report 1998/6, Dept. of Mathematics and Computer Science, University of Leicester, 1998.

[LR97] G. Lowe and A. W. Roscoe. Using CSP to detect errors in the TMN protocol. *Software Engineering*, 23(10):659–669, 1997.

[MC02] G. Montenegro and C. Castelluccia. Statistically Unique and Cryptographically Verifiable (SUCV) identifiers and addresses. In *Network and Distributed Systems Security Symposium*. Internet Society, February 2002.

[Nik01] Pekka Nikander. Denial-of-service, address ownership, and early authentication in the IPv6 world. In B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, editors, *Security Protocols*, number 2467 in Lecture Notes in Computer Science. Springer, 2001.

[NS78a] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12), December 1978.

[NS78b] Roger Needham and Michael Schroeder. Using encryption for authentification in large networks of computers. *Communications of the ACM*, 21(12), December 1978.

[NS87] R. Needham and M. Schroeder. Authentication revisited. *Operating Systems Review*, 21(7), January 1987.

[NS93] B. Clifford Neumann and Stuart G. Stubblebine. A note on the use of timestamps as nonces. *Operating Systems Review*, 27(2):10–14, april 1993.

[NT94] B. Clifford Neuman and Theodore Ts'o. Kerberos : An authentication service for computer networks. Technical Report ISI/RS-94-399, USC/ISI, 1994.

[NYW03] Pekka Nikander, Yukka Ylitalo, and Jorma Wall. Integrating security, mobility and multi-homing in a HIP way. In *Network and Distributed Systems Security Symposium*, 2003.

[OR87] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.

[OR01] Greg O'Shea and Michael Roe. Child-proof authentication for MIPv6 (CAM). *Computer Communications Review*, April 2001.

[Pau01] Lawrence C. Paulson. Relations between secrets: Two formal analyses of the yahalom protocol. *J. Computer Security*, 2001.

[RAOA02] M. Roe, T. Aura, G. O'Shea, and J. Arkko. Authentication of mobile IPv6 binding updates and acknowledgments. Internet draft, February 2002.

[RS98] P. Y. A. Ryan and S. A. Schneider. An attack on a recursive authentication protocol: A cautionary tale. *Information Processing Letters*, 65(1):7–10, 1998.

[Sat89] M. Satyanarayanan. Integrating security in a large distributed system. *ACM Transactions on Computer Systems*, 7(3):247–280, 1989.

[Sho96] Victor Shoup. A note on session key distribution using smart cards. http://www.shoup.net/papers/update.ps, july 1996.

[Sim88] Gustavus J. Simmons. An impersonation-proof identity verification scheme. In *Advances in Cryptology: Proceedings of Crypto 87*, volume 293 of *LNCS*, pages 211–215. Springer-Verlag, 1988.

[Sim94] Gustavus J. Simmons. Cryptoanalysis and protocol failure. *Communications of the ACM*, 37(11):56–65, November 1994.

[SM01] Vitaly Shmatikov and John Mitchell. Finite-state analysis of two contract signing protocols. *Special issue of Theoretical Computer Science on security*, 2001. Accepted for publication.

[SMB90] Michael Merritt Steven M. Bellovin. Limitations of the kerberos authentication system. *Computer Communication Review*, 20(5):119–132, october 1990.

[SR96] Victor Shoup and Avi Rubin. Session key distribution using smart cards. In *In Proceedings of Advances in Cryptology, EUROCRYPT'96*, volume 1070 of *LNCS*. Springer-Verlag, 1996.

[Syv94] Paul Syverson. A taxonomy of replay attacks. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 131–136. IEEE Computer Society Press, 1994.

[Tho01] Thomson. Smartright technical white paper v1.0. Technical report, Thomson, october 2001. http://www.smartright.org.

[TMN89] M. Tatebayashi, N. Matsuzaki, and D.B. Newman. Key distribution protocol for digital mobile communication systems. In *Advance in Cryptology — CRYPTO '89*, volume 435 of *LNCS*, pages 324–333. Springer-Verlag, 1989.

[Wei99] Christoph Weidenbach. Towards an automatic analysis of security protocols. In Harald Ganzinger, editor, *Proceedings of the*

*16th International Conference on Automated Deduction*, volume 1632 of *LNAI*, pages 378–382. Springer, 1999.

[WL94] T. Y. C. Woo and S. S. Lam. A lesson on authentication protocol design. *Operating Systems Review*, 1994.

[YOM91] Suguru Yamaguchi, Kiyohiko Okayama, and Hideo Miyahara. The design and implementation of an authentication system for the wide area distributed environment. *IEICE Transactions on Information and Systems*, E74(11):3902–3909, November 1991.