# Establishing and Fixing a Freshness Flaw in a Key-Distribution and Authentication Protocol

Reiner Dojen*, Ioana Lasc*, Tom Coffey*
*Department of Electronic & Computer Engineering, University of Limerick, Ireland.
E-Mail: reiner.dojen@ul.ie, ioana.lasc@ul.ie, tom.coffey@ul.ie

## Abstract

*The security of electronic networks and information systems is nowadays seen as a critical issue for the growth of Information and Communication Technologies. Cryptographic protocols are used to provide security services such as confidentiality, message integrity, authentication, certified E-mail and non-repudiation. Traditionally, security protocols have been designed and verified using informal techniques. However, the absence of formal verification can lead to security errors remaining undetected. Formal verification techniques provide a systematic way of discovering protocol flaws.*

*This paper establishes a freshness flaw in a key-distribution and Authentication Protocol using an automated logic-based verification engine. The performed verification reveals a freshness flaw in the protocol that allows an intruder to impersonate legitimate principals. The cause of the freshness flaw is discussed and an amended protocol is proposed. Formal verification of the amended protocol provides confidence in the correctness and effectiveness of the proposed modifications.*

## 1. Introduction

The security of electronic networks and information systems is nowadays seen as a critical issue for the growth of Information and Communication Technologies. Cryptographic security protocols are widely used for the secure delivery of data over mobile and fixed networks. As these networks are trusted with highly sensitive information, these security protocols are required to ensure the security of both the infrastructure itself and the information that runs through it. These protocols can be thought of as the keystones of a secure architecture. Basic cryptographic protocols allow agents to authenticate each other, to establish fresh session keys for confidential communication and to ensure the authenticity of data and services. Building on such basic cryptographic protocols, more advanced services like non-repudiation, fairness, electronic payment and electronic contract signing are achieved. As these security protocols form the backbone of applications that involve the transmission of sensitive data, it is critical that the correctness of the employed security protocols can be formally proven. However, the design of provably correct security protocols is complex and highly prone to error. The main difficulty in the development of security protocols is to address the vast possibilities of an adversary to gain information [1]: Further, it has to be assumed that the network is hostile and under the complete control of the adversary.

Formal verification of a security protocol is an essential part of the design process [2], as it:

- provides a systematic way to detect design flaws
- identifies the exact cryptographic porperties a protocol aims to satisfy
- identifies the assumptions and the environment under which these properties hold
- removes ambiguity in specifications of protocols.

Often, only informal and intuitive techniques are used to analyse security protocols. This has resulted in the use of insecure protocols in the public domain. On the other hand, formal verification techniques have successfully identified several previously unknown flaws in security protocols [2, 3, 4, 5, 6, 7, 8, 9].

This paper concerns the verification of security protocols using a logic-based proving engine. Section II discusses security protocols and different approaches to their formal verification and Section III introduces the CDVT verification engine. In Section IV the Kao-Chow key distribution and authentication protocol v1, v2 and v3 are discussed. The formal verification of the Kao Chow protocol is detailed in Section V and an attack that allows an intruder to impersonate legitimate principals is presented. In Section VI amendments to the Kao-Chow protocol are proposed and Section VII presents a formal verification of the amended protocol. Finally, Section VIII concludes this paper.

## 2. Security protocols and their verification

Security protocols are used to ensure secure communications over insecure networks.

Authentication is a major concern when establishing such secure communications. Authentication includes *entity authentication* and *data origin authentication*. *Identification* or *entity authentication* is achieved through mechanisms that provide one party of the communication process of the second's identity and the fact that the second was active at the time it sent data to the first. *Data origin authentication* techniques provide to the party that receives a message evidence about the sender's identity.

*Data freshness* is an essential security requirement: the content of the messages exchanged between the parties should belong to the current session or protocol run. In other words, the data generated for a certain session should not be valid or accepted by the principals in any other session of the protocol. This ensures that an attacker who gains access to a message or parts of a message cannot use it in a *replay attack*.

*A replay attack* on a cryptographic protocol is a breach of security that enables an intruder to record information and send it to honest principals during the current run of the protocol or in a future one.

In a *freshness attack* the intruder replays recorded data that is either modified or sent to an other principal than intended by the original sender. This type of attack makes use of the lack of freshness identifiers in the exchanged messages or the non-specific format of the messages that a principal sends out and receives.

A *parallel session attack* is a type of replay attack, where another session of the same protocol starts before the initial one is completed. The intruder uses messages or message components from one session to synthesise messages in the other session.

Formal verification is an essential step to confirm the security and effetiveness of cryptographic protocols. Several techniques can be used to analyze the security of protocols. These include state machines/model checkers and modal logics [10].

### 2.1. State Machines

State machines can be used to analyze protocols using a method known as the reachability analysis technique [11]. Using the technique, the global state of the system is expressed for each transition. Each global state is then analyzed - if a state is discovered where an attacker gains access to information which should be secret, there is a problem with the protocol. An exhaustive search checks that all reachable states are safe. Reachability analyses are suitable for determining if a protocol is correct with respect to its specification but they do not guarantee security from an active attacker. Another serious limitation of this technique is that drastic assumptions are required in order to keep the state-space small [12].

Several tools are available for state-space analysis, e.g. Murφ [13], FDR [14], the NRL Protocol Analyser [15], Brutus [16], Athena [17] and AVISPA [18].

### 2.2. Modal Logics

Logic-based formal verification of cryptographic security protocols proves the correctness of the protocol against its goals. The technique of logic-based formal verification is accredited largely to Burrow, Abadi and Needham, developers of the BAN logic [3]. This work initiated intense research in the area of logic-based formal verification and several logics, such as GNY [4], SVO[19], CS [20] and ZV [6] have been developed on the basis of BAN.

Logic-based verification of cryptographic protocols involves a process of deductive reasoning, where the desired protocol goals are deduced from the formalised protocol. The first steps in logic-based verification involves specifying the protocol steps, the initial assumptions and the protocol goals in the language of the logic. The final verification step concerns the application of logical postulates to establish the beliefs and possessions of protocol principals. The objective of the logical analysis is to verify whether the desired goals of the protocol can be derived from the initial assumptions and protocol steps. If such a derivation exists, the protocol is successfully verified; otherwise, the verification fails. A successfully verified protocol can be considered secure within the scope of the logic. On the other hand, the results of a failed verification are helpful, as they point to missing assumptions or weaknesses in the protocol. If a weakness is discovered, the protocol should be redesigned and re-verified.

## 3. CDVT - An automated logic-based verification engine

The CDVT verification engine (cf. Fig. 1) is an automated system that implements a modal logic of knowledge and belief using Layered Proving Trees [1]. The implemented logic can analyze the evolution of both knowledge and belief during a protocol execution and is therefore useful in addressing issues of both security and trust.

### 3.1. The language of the CDVT verification engine

The CDVT verification engine uses a parser to read in the protocol specification from a text file. Table 2 summarises the atomic units of the textual grammar.

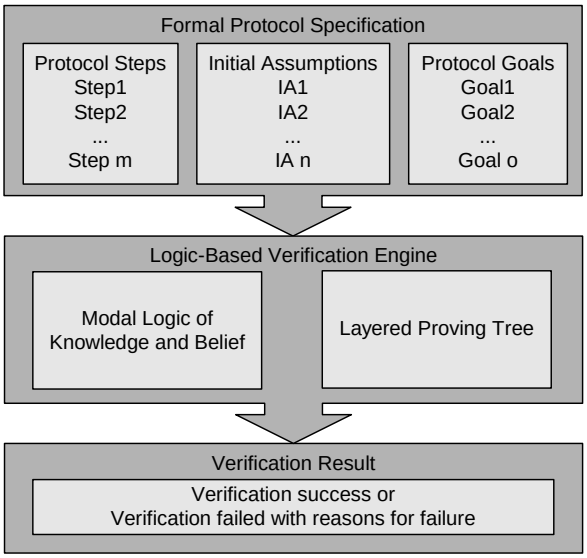


**Figure 1: The CDVT verification engine**

**Table 1: Atomic units of textual grammar**

| Textual Grammar | Regular Expression |
|---|---|
| Principal | [AB-EIJLMOQRSU-Z][A-Za-z_0-9_]* |
| Trusted Principal | TTP[A-Za-z0-9_]* |
| Sym. Key | K[a-z][a-zA-Z0-9_]* |
| Public Key | K[a-z][A-Za-z0-9_]*Pub |
| Private Key | K[a-z][A-Za-z0-9_]*Priv |
| Nonce | N[a-z][A-Za-z0-9_]* |
| Timestamp | TS[a-z][A-Za-z0-9_]* |
| Function | F[A-Za-z0-9_]* |
| Hash | H[A-Za-z0-9_]* |
| Binary Data | [a-z][A-Za-z0-9_]* |

Composite data components are constructed according to Table 2, where elements follow the regular expressions as given in Table 1 and *Data* represents an arbitrary data element (either atomic unit or composite data).

**Table 2: Composite data construction**

| Composite Data | Textual Representation |
|---|---|
| Concatenation | *Data,Data* |
| Group Element | ( *Data* ) |
| Symmetric Encryption | {*Data*}*Data* |
| Public Key Encryption | {*Data*}KPub |
| Private Key Encryption | {*Data*}KPriv |
| Function of Data | F(*Data*) |
| Hash of Data | H(*Data*) |

Statements are defined as by the rules presented in Table 3, where elements follow the regular expressions as given in Table 1, *Data* is either an atomic data unit or a composite data as defined in Table 2, *operator* is any of "send", "receive" or "possess", "i" indicates the indexed discrete time and *Statement* represents an arbitrary statement.

**Table 3: Statement construction**

| |
|---|
| Principal *operator* at[i] *Data* |
| Principal know at[i] *Statement* |
| Principal believe at[i] *Statement* |
| Principal know at[i] NOT ( *Statement* ) |
| Principal believe at[i] NOT ( *Statement* ) |
| ( *Statement* ) |
| ( *Statement* AND *Statement* ) |
| ( *Statement* IMPLY *Statement* ) |

Each line of the textual specification file is preceded by a label. Assumptions are labeled "An", protocol steps are labelled "Sn" and protocol goals are labelled "Gn", where n numbers each group sequentially. Every line must be closed with a semicolon (';') and comments are introduced by a double forward slash ('//' – C++ style comments).

The inference rules provided are the standard rules of natural deduction. The axioms of the logic express the fundamental properties of public-key cryptographic protocols such as the ability of a principal to encrypt/decrypt based on knowledge of a cryptographic key. The axioms also reflect the underlying assumptions of the logic, which are as follows:

- The communication environment is reliable, but hostile. That is, message loss or modification can only occur as consequence of hostile intervention.
- The cryptosystem is ideal. That is, the encryption and decryption functions are completely non-invertible without knowledge of the appropriate cryptographic key and are invertible with knowledge of the appropriate cryptographic key.
- A public key used by the system is considered valid if it has not exceeded its validity period and only its rightful owner knows the corresponding secret key.
- If a piece of data is encrypted/decrypted, then the entity which performed the encryption/decryption must know that data.

## 4. The Kao-Chow key distribution and mutual authentication protocol

The Kao-Chow protocol [21] is a mutual authentication and key distribution protocol aiming at strong authentication and low message overhead. It uses a third trusted party - the server S – for key generation and distribution. The two parties that want to communicate securely trust S to issue a fresh secret session key Kab. A and B will use this key to encrypt future message exchanges. Both principals send and receive messages to/from the server encrypted with the

long term symmetric keys Kas and Kbs. Further, the protocol aims to authenticate the principals A and B to each other using their nonces Na and Nb. The following notations are used throughout this paper:

- A, B, S: principals
- Kas, Kbs: symmetric keys shared by A/S and B/S
- Kab: fresh session key generated by S
- Na, Nb: nonces (once off random numbers) of A and B used in the authentication process
- {X}K: encryption of data X with the key K
- A -> B: Msg: Principal A sends Msg to B.

The following outlines the informal description of the Kao-Chow protocol v1:

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

In the first message A sends to the server his identity, a fresh nonce Na and B's identity. In the next step server S will issue two tickets that contain the session key to B. Both tickets consist of the identities of A and B, the nonce Na and the session key Kab. The first ticket is encrypted with Kas and the second is encrypted with the Kbs. As B can decrypt the second ticket it now is in possession of the session key. In step three B will forward the first ticket from the second message along with the nonce Na encrypted under Kab and its own nonce Nb to A. On receipt of this message A retrieves the session key Kab and verifies the correctness of the encryped nonce. In the final step A will encrypt the nonce Nb with Kab and sends it to B.

The authors discuss limitations of Kao-Chow v1 and proposed another two versions (Kao-Chow v2 & Kao-Chow v3) for this protocol. Kao-Chow v2 aims to overcome the limitations in the first version, by adding an extra fresh key Kt generated by the sever that is discarded after each protocol run. Kt is included in message two when the distribution of Kab is initiated:

*Kao-Chow v2*

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
4. A -> B: {Nb, Kab}Kt

The third version of the protocol is an extension of v2 to encompass tickets. In step three B generates a new ticket containing Kab and a timestamp Ta:

*Kao-Chow v3*

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

## 5. Verification of the Kao-Chow key distribution and authentication protocol

This section uses the CDVT verification engine to establish the correctness of the Kao-Chow protocol. In order to apply CDVT, the protocol must be formalised, i.e. translated into the language of the tool. A formalised protocol consists of three components:

- Initial assumptions
- Protocol steps
- Protocol goals

The verification engine then applies the postulates of the implemented logic in an attempt to derive the protocol goals as a logical consequence of the initial assumptions and the protocol steps. If such a derivation exists, the verification is successful and the verified protocol can be considered secure within the scope of the logic. If the verification fails, investigation of the verification process can point to missing assumptions or weaknesses. If a weakness is discovered, the protocol should be re-designed and re-verified.

### 5.1. Initial assumptions

The following specifiec the initial assumptions of the Kao-Chow protocol:

A1: A possess at[0] Kas;
A2: S possess at[0] Kas;
A3: A know at[0] S possess at[0] Kas;
A4: B possess at[0] Kbs;
A5: S possess at[0] Kbs;
A6: B know at[0] S possess at[0] Kbs;
A7: A possess at[0] Na;
A8: A know at[0] NOT(Zero possess at[0] Na);
A9: B possess at[0] Nb;
A10: B know at[0] NOT (Zero possess at[0] Nb);
A11: S possess at[0] Kab;
A12: S know at[0] NOT (Zero possess at[0] Kab);
A13: B know at[0] (B know at[2] S send at[2] {A,B,Na,Kab}Kbs AND B know at[2] NOT(Zero send at[0] {A,B,Na,Kab}Kbs ) ) IMPLY A possess at[4] Kab);
A14: A know at[0] (A know at[3] S send at[3] {A,B,Na,Kab}Kas AND A know at[3] NOT(Zero send at[0] {A,B,Na,Kab}Kbs )) IMPLY B possess at[3] Kab);

A1 expresses the fact that before the start of the protocol run the principal A possess the key and in a similar way, A2 expresses the fact that the server S possesses the key Kas at the same time. A3 specifies that before the start of the protocol run, A is aware of the fact that S possesses the key Kas. With these three assumptions it is specified that A and S share the secret

key Kas. The assumptions A4-A6 describe in the same manner the possession of the key Kbs by B and S.

Assumptions A7 expresses the belief that the principal A possesses the nonce Na and the assumption A8 states that A knows before the start of the protocol that no other principal possesses this nonce at that time. Assumptions A9 and A10 specify the corresponding knowledge of B on the nonce Nb.

Assumption A11 specifies that the fresh session key Kab is possessed by the server S and A12 indicates that the server knows that it is the only principal that possess this key before the start of the protocol run.

Finally, assumption A13 states that if principal B can deduce that the message {A,B,Na,Kab}Kbs has been send by S and also that it has been send during the current protocol run, then B can also deduce that Kab is a good session key shared with A and that A possesses this session key. Assumption A14 states the corresponding information for A.

### 5.2. Kao-Chow v1 protocol steps

The steps of the Kao-Chow protocol v1 are formalised as follows:

S1: S receive at[1] A,B,Na;
S2: B receive at[2] {A,B,Na,Kab}Kas,{A,B,Na,Kab}Kbs;
S3: A receive at[3] {A,B,Na,Kab}Kas,{Na}Kab,Nb;
S4: B receive at[4] {Nb}Kab;

### 5.3. Kao-Chow v1 protocol goals

In the goals section the objectives of the protocol are specified. In this case, the objectives are the mutual authentication of A and B and distribution and secrecy of the session key Kab. These goals are formalized as follows:

G1: B possess at[2] Kab;
G2: B know at[2] S send at[2] {A,B,Na,Kab}Kbs;
G3: B know at[2] NOT(Zero send at[0] {A,B,Na,Kab}Kbs);
G4: A possess at[3] Kab;
G5: A know at[3] S send at[3] {A,B,Na,Kab}Kas;
G6: A know at[3] NOT(Zero send at[0] {A,B,Na,Kab}Kas);
G7: A know at[3] B send at[3] {Na}Kab;
G8: A know at[3] NOT(Zero send at[0] {Na}Kab);
G9: B know at[4] A send at[4] {Nb}Kab;
G10: B know at[4] NOT(Zero send at[0] {Nb}Kab);

Goals G1-G3 correspond to key establishment for B: G1 states that B possesses the session key Kab after step 2 completes. G2 states that B knows that the message component containing the session key originates at the server S and G3 states that B knows that this message component is fresh, i.e. that it has been created by the server for the current protocol run. Goals G4-G6 are the corresponding key establishment goals for A after step 3 completes.

Goals G7-G8 relate to authentication of B to A. G7 states that A knows that B is indeed the source of message component {Na}Kab, i.e. of the reply to A's nonce challenge. G8 states that A knows that this message component has been created during the current protocol run. Goals G9-G10 are corresponding goals regarding authentication of A to B.

### 5.4. Kao-Chow v1 verification results

The results of the automated verification are presented in Fig. 2. It can be seen, that not all goals of key establishment for B (goals 1-3) and authentication of A to B (goals 9-10) are satisfied. On the other hand, the goals concerning key estabishment for A (goals 4-6) and authentication of B to A (goals 7-8) are verified successfully.

The verification tool allows to investigate the reasons for failed goals by browsing the details of the verification process. For example, Fig. 3 details the failed verification of goal G9, where it can be seen that B's inability to establish A's possesson of the session key Kab is the reason for the failure.

Investigation of the failed protocol goals in this fashion, reveals that the protocol suffers from a freshness weakness: B's inability to establish freshness of the message component containing the session key in protocol step 2 (goal G3) prevents B to accept the session key. As a consequence, goal G9 also fails. Thus neither key establishment for B nor authentication of A to B is achieved by the protocol.

### 5.5. An attack on the Kao-Chow protocol v1

The discovered weakness of the protocol can be exploited by an intruder. A disonest party could record the messages exchanged in a previous session of this protocol and cause the use of an old session key. Under the assumption that the old session key is compromised, the intruder can even impersonate A to B. Such an attack is performed as follows: In this attack, protocol step 1 is omitted. Instead, the intruder will send an old recorded protocol step 2 that contains a compromised session key Kab* to B. B assumes that this message is a legitimate message from the server and assumes that principal A wishes to initiate communication. Note that B is not able to detect that this message is a replay. Thus, B will retrieve the compromised session key Kab* and will continue with the protocol. The intruder intercepts any following

messages originating from B. Since session key Kab* is compromised, the intruder is able to reply correctly to B's nonce challenge by sending {Nb}Kab* to B. At the end of the protocol B is convinced to have established a new valid session with principal A, when in fact, B communicates with an intruder and A has no knowledge of this session.

### 5.6. Comments on the security of Kao-Chow v2 and v3

Version 2 of the Kao-Chow protocol uses a fresh key Kt to avoid freshness weaknesses. However, investigation of the properties of the Kt reveal that it does not add to the security of the protocol. If Kt is a confirmed shared fresh secret between A and B, then there is no need to establish another session key Kab, as A and B could simply use Kt for communication. On the other hand, if the key Kt is not a confirmed shared fresh secret but would be stored after each protocol run and the Kt produced in futures runs would be compared to the old Kts the replay would be impossible. However, the authors introduced this key with the condition that it is discarded after each session. Therefore it has exactly the same properties as Kab, which makes it redundant

The addition of tickets in version 3 of the Kao-Chow protocol will not prevent the above attack, as it is generated by B in step 3. At this time, the intruder has already compromised the session and is able to impersonate A to B.

Verification of the Kao-Chow protocols v2 and v3 using CDVT confirms that neither version 2 nor version 3 are able to prevent the presented attack.

## 6. Amending the Kao-Chow protocol

As outlined in the previous section, neither v2 nor v3 of the Kao-Chow protocol can be considered secure. The authors argue that addition of a nonce handshake to their first version of the protocol will add at lest two extra messages to the protocol. However, their objective was to keep the number of messages to a minimum. Further, they rule out the use of timestamps, as it would require clocks synchronization with all the inherent problems. In order to prevent the presented replay attack, both parties are required to contribute a fresh component to the messages. Therefore, it is impossible to perform the protocol in four steps. However, secure authentication and key agreement can be achieved in five steps, i.e. with one extra message exchange. Further, the extra message will not contain any enryption and, therefore, will keep the extra burden to a minimum.

The following proposes amendmends to the Kao-Chow key-distribution and authentication protocol:

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

Rather than sending the request for communication directly to the server, the initiating principal A will send the request along with the nonce Na to the principal B. The Responder B then will forward the communication request to the server along with Na and its own nonce Nb. Thus, the tickets that contain the session key can be identified by both principals as fresh, i.e. as belonging to the current protocol run. Consequently, any attempt by an intruder to replay message 3 will fail, as B can identify the replay through the wrong value of Nb.

Another change has been made for optimization in step three. The ticket issued by the server for A includes only B's identity and A's nonce rather than having both principals' identities and nonces. Similarily, the ticket sent to B contains only A's identity and B's nonce.

## 7. Verification of the proposed amended version of the Kao-Chow protocol

### 7.1. Initial assumptions

As no data was added to the protocol, the assumptions are essentially identical with the ones for the Kao Chow V1 protocol. The only changes occur in assumptions A13 and A14, where the modification to the messages are reflected:

A1: A possess at[0] Kas;
A2: S possess at[0] Kas;
A3: A know at[0] S possess at[0] Kas;
A4: B possess at[0] Kbs;
A5: S possess at[0] Kbs;
A6: B know at[0] S possess at[0] Kbs;
A7: A possess at[0] Na;
A8: A know at[0] NOT(Zero possess at[0] Na);
A9: B possess at[0] Nb;
A10: B know at[0] NOT (Zero possess at[0] Nb);
A11: S possess at[0] Kab;
A12: S know at[0] NOT (Zero possess at[0] Kab);
A13: B know at[0] (
(B know at[3] S send at[3] {A,Nb,Kab}Kbs AND
B know at[3] NOT(Zero send at[0]
{A,Nb,Kab}Kbs ) ) IMPLY A possess at[3] Kab);
A14: A know at[0] (
(A know at[4] S send at[4] {B,Na,Kab}Kas AND
A know at[4] NOT(Zero send at[0]
{B,Na,Kab}Kbs)) IMPLY B possess at[4] Kab);

### 7.2. Protocol steps

Having changed the steps of the protocol, the formalisation needs to be adjusted as follows:
S1: B receive at[1] A, Na;
S2: S receive at[2] A, B, Na, Nb;
S3: B receive at[3] {B,Na,Kab}Kas, {A,Nb, Kab}Kbs;
S4: A receive at[4] {B, Na, Kab}Kas, {Na, Nb}Kab;
S5: B receive at[5] {Nb}Kab;

### 7.3. Protocol goals

Due to the inclusion of the second message, the key distribution will start one step later than in the original protocol. Further, the modifications of the steps are also reflected in the corresponding goals:
G1: B possess at[3] Kab;
G2: B know at[3] S send at[3] {A,Nb,Kab}Kbs;
G3: B know at[3] NOT(Zero send at[0] {A,Nb,Kab}Kbs);
G4: A possess at[4] Kab;
G5: A know at[4] S send at[4] {B,Na,Kab}Kas;
G6: A know at[4] NOT(Zero send at[0] {B,Na,Kab}Kas);
G7: A know at[4] B send at[4] {Na,Nb}Kab;
G8: A know at[4] NOT(Zero send at[0] {Na,Nb}Kab);
G9: B know at[5] A send at[5] {Nb}Kab;
G10: B know at[5] NOT(Zero send at[0] {Nb}Kab);

### 7.4. Results of the verification

Verifying the amended version of the protocol will produce the results shown in Fig. 4. where all goals are verified successfully.

This provides confidence in the correctness of the proposed modifications

## 8. Conclusions

This paper discussed the formal verification of security protocols using techniques based on state-space machines and modal logics. The automated logic-based verification engine CDVT was presented and its input language was detailed.

As an example, the Kao-Chow key distribution and authentication protocol was analysed. Details of the formalised Kao-Chow protocol were presented and the results were discussed. The performed verification proves that the Kao-Chow protocol fails to achieve all the required goals. Investigation of the verification details reveals a freshness flaw in the protocol. This flaw can be exploited by an intruder to impersonate legitimate principals. Discussion on versions 2 and 3 of the Kao-Chow protocol demonstrates they contain the same weakness.

The cause of the freshness flaw in the Kao-Chow protocol was discussed and an amended protocol was proposed. Formal verification of the amended protocol provides confidence in the correctness and effectiveness of the proposed modifications

## 9. Acknowledgements

This work was funded by the Irish Research Council for Science, Engineering and Technology (IRCSET Embark Initiative) and Science Foundation Ireland - Research Frontiers Programme(SFI RFP07CMSF 631).

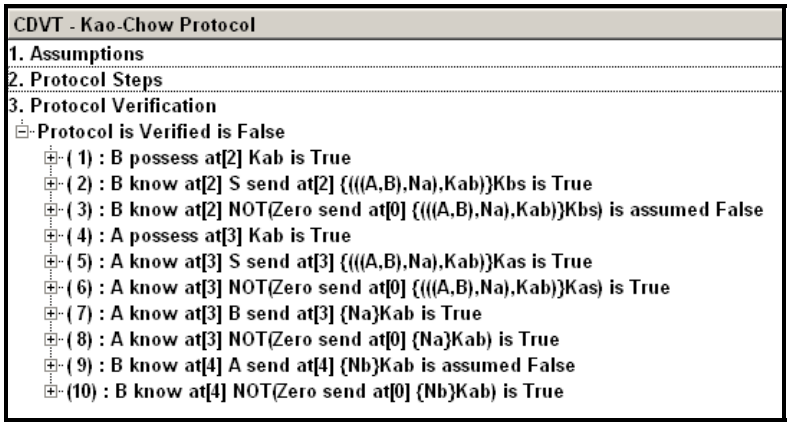


**Figure 2: Kao-Chow v1 verification results**

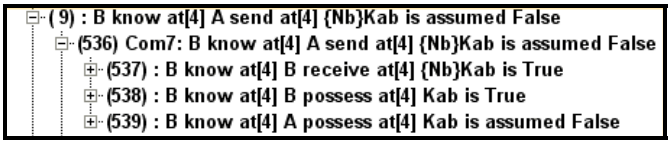


**Figure 3: Investigation of goal 9 failure**

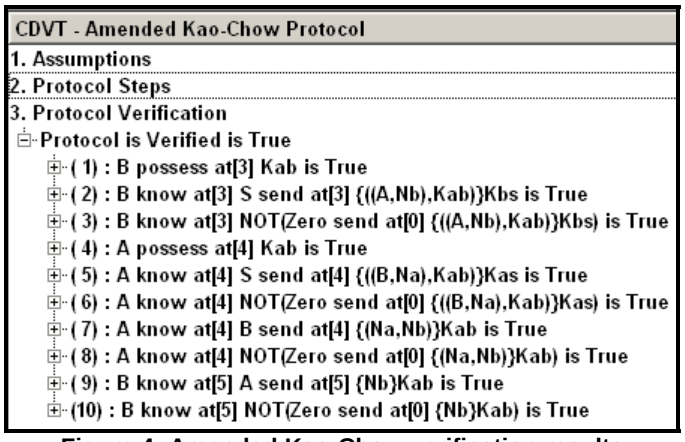


**Figure 4: Amended Kao-Chow verification results**

## 10. References


[1] R. Dojen, and T. Coffey, Layered Proving Trees: A Novel Approach to the Automation of Logic-Based Security Protocol Verification, ACM Transactions on Information and System Security (TISSEC), Vol.8, Issue 3, August 2005, pp. 287-311.

[2] T. Coffey, R. Dojen and T. Flanagan, Formal verification: An imperative step in the design of security protocols, Computer Networks Journal 43:5 (2003) 601–618

[3] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. ACM Transactions on Computer Systems, 8 February 1990.

[4] L, Gong, R. Needham and R. Yahalom. Reasoning about belief in cryptographic protocols. Proceedings of the IEEE Computer Society Symposuim on Security and Privacy, pages 234-248, May 1990.

[5] G. Bella, F. Massacci, L.C. Paulson, and P. Tramontano: Formal verification of cardholder registration in SET. *European Symposium on Research in Computer Security Computer - ESORICS 2000*, Toulouse, France, October 2000, pp. 159–174

[6] Y. Zhang and V. Varadharajan: A logic for modelling the dynamics of beliefs in cryptographic protocols. Australasian Computer Science Conference, Gold Coast, Queensland, Australia, Feb. 2001, pp. 215-222.

[7] S. Gürgens and C. Rudolph, Security Analysis of (Un-) Fair Non-repudiation Protocols. Formal Aspects of Security, LNCS, vol. 2629, Springer-Verlag, London, (2002) 97-114

[8] T. Coffey, M. Ventuneac, T. Newe and I. Salomie, On investigating the security and fairness of a fair exchange protocol using logic-based verification, *IEEE International Conference on Intelligent Engineering Systems (INES2004)*, Cluj-Napoca, Romania, September 2004, 325-330.

[9] M. Ventuneac, R. Dojen and T. Coffey, Automated Verification of Wireless Security Protocols using Layered Proving Trees, WSEAS Transactions on Communications, 5:2 (2006) 252-258

[10] T. Coffey, R. Dojen and T. Flanagan, On Different Approaches to Establish the Security of Cryptographic Protocols, *Proceedings of SAM'03*, Vol. II, Las Vegas, USA, June 2003, 637-643

[11] C.H. West: General Techniques for Communications Protocol Validation. IBM Journal of Research and Development, Vol. 22, No. 4, 1978, pp. 393-404

[12] L.C. Paulson: The Inductive Approach to Verifying Cryptographic Protocols, *Journal of Computer Security*, Vol. 6, 1998, pp. 85-128

[13] D.L. Dill: The Murφ verification system. *International Conference of Computer Aided Verification*, New Brunswick, NJ, USA, July 1996, pp 390-393.

[14] G. Lowe: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Software - Concepts and Tools*, Volume 17, No. 93, 1996, pp. 93-102.

[15] C. Meadows: The NRL Protocol Analyser: an overview. *Journal of Logic Programming,* 26(8), 1996, pp. 113-131.

[16] E.M. Clarke, S. Jha and W. Marrero: Verifying security protocols with Brutus. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 9, No. 4, 2000, pp.443-487.

[17] D. Song, S. Berezin and A. Perrig.: Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, Vol.9, No.1, 2001, pp. 47–74

[18] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò and L. Vigneron: The Avispa Tool for the automated validation of internet security protocols and applications. Proceedings of CAV2005, Computer Aided Verification, LNCS 3576, Springer Verlag.

[19] P.F. Syverson and P.C. van Oorschot: On unifying some cryptographic protocols logics. *IEEE Symposium on Security and Privacy*, Oakland, USA, May 1994, pp.14-28.

[20] T. Coffey, and P. Saidha, A Logic for Verifying Public-Key Cryptographic Protocols. IEEE Proceedings of Computers and Digital Techniques, Vol.144, No.1, 1997, pp. 28-32.

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