

Finite State Analysis of IKE

by: Udayan Kumar, Y.Narendra, and Rakesh Upadhyay, 05/30/2005

<http://www.securitydocs.com/library/3333>

IP Security Protocol (IPSEC) Working Group of the Internet Engineering Task Force (IETF) developed the Internet Key Exchange protocol (IKE) [1], a key exchange protocol. The main purpose of its development is to provide the security support for client protocols of the Internet Protocol. It is being designed to do much more than just distribute keys, it can also be used to establish Security Associations that specify such things as the protocol format used, the cryptographic and hashing algorithms used, and other necessary features for secure communication. For incorporating flexibility, it supports a number of different types of key exchange options, including digital signatures, public key encryption, and conventional encryption using shared keys. The Diffie-Hellman algorithm is used to generate shared key material.

The IKE protocol has evolved from ISAKMP [2] and Oakley [3]. It can be thought of as combination of these two protocols. ISAKMP provides a framework for establishing security associations and cryptographic keys, but does not prescribe any particular authentication mechanism. Indeed, ISAKMP is supposed to integrate with a number of security protocols. Oakley, on the other hand, is a suite of key agreement protocols in which two parties generate a key jointly.

A typical key establishment protocol proceeds in one phase, in which two parties use master keys to establish shared keying material. IKE, however, proceeds in two such phases. In the first phase, two entities use master keys to agree, not only on keying material, but on the various mechanisms (e.g. cryptographic algorithms, hash functions, etc.), that they will use in the second phase. The keying material and set of mechanisms thus agreed upon is called a security association. In the second phase, the keys and mechanisms produced in the first phase are used to agree upon new keys and mechanisms (that is, new security associations) that will be used to protect and authenticate further communications. The security association established in Phase One is bidirectional, so the initiator in the first phase can be either initiator or responder in the second phase. Like Oakley, IKE can also be used in multiple modes, aggressive mode and main mode.

The paper step by step analyzes the threat perception and measures taken to address them, when we build IKE protocol from the scratch. We have used Murf tool, to check the possibilities of attack on the strip down versions of IKE protocol in Public signature, Pre-Shared secret and Public Encryption mode. For removing the threats we added some more steps, which finally lead us to the original IKE protocols.

Now we need to have some understanding of the Formal methods, its importance and also the difficulties in modeling the protocols. Formal verification of a protocol proceeds by describing the protocol in some language and then comparing the behavior of this description with a specification of the desired behavior. A verifier generates states from the description, comparing them with the specification as it goes. If the verifier detects an inconsistency, this fact is reported, along with an example sequence of states that illustrates how the problem can occur, that aid in diagnosis.

The main challenges that arise while using finite state verification tool for analyzing security-related protocols are:

- State-space explosion
- Subtleties involving formalization of the adversary and
- Subtleties involving the properties of the encryption primitives, which may be modeled as completely secure black-box primitives, or primitives with other algebraic properties.

Outline of the methodology

The Murf verification system

Murf is a protocol verification tool that has been successfully applied to several industrial protocols, especially in the domains

of multiprocessor cache coherence protocols and multiprocessor memory models. Previously this tool has been successfully used to model and verify SSL 3.0 [10], Needham-Schroeder public key protocol Kerberos and the TMN cellular telephone protocol.

To use Murf for verification, one has to model the protocol in the Murf language and augment this model with a specification of the desired properties [8]. The Murf system automatically checks, by explicit state enumeration, if all reachable states of the model satisfy the given specification. For the state enumeration, either breadth-first or depth-first search can be selected.

The state of the model consists of the values of all global variables. In a startstate, initial values are assigned to global variables. The transition from one state to another is performed by rules. Each rule has a Boolean condition and an action, which is a program segment that is executed atomically. The action may be executed if the condition is true (i.e. the rule is enabled) and may change global variables.

The desired properties of a protocol can be specified in Murf by invariants, which are Boolean conditions that have to be true in every reachable state. If a state is reached in which some invariant is violated, Murf prints an error trace – a sequence of states from the start state to the state exhibiting the problem.

The methodology

We have analyzed the protocols using the following sequence of steps:

1. Formulate the protocol. This involves simplifying the protocol by identifying the key steps and primitives. However, the Murf formulation of a protocol is more detailed than the high level description found in the literature. The most significant issue is to decide exactly which messages will be accepted by each participant in the protocol.
2. Add an adversary to the system. We generally assume that the adversary is a participant in the system capable of initiating communication with an honest participant. We also assume that the network is under the control of the adversary and allow the adversary the following actions:
3. Overhear every message, remember all parts of each message, and decrypt ciphertext when it has the key.
4. Intercept (delete) messages.
5. Generate messages using any combination of initial knowledge about the system and parts of overheard messages.
6. State the desired correctness condition.
7. Run the protocol for some specific choice of system size parameters.
8. Experiment with alternate formulations and repeat.

Murf compiler and verifier

The Murf compiler takes a Murf source description and generates a C++ program, which is compiled together with code for a verifier which checks for invariant violations, error statements and deadlocks.

The intruder model

The intruder model has the following limitations.

1. No cryptanalysis Here the intruder ignores both computational and numbertheoretic properties of cryptographic functions. As a result it cannot perform any cryptanalysis whatsoever.
2. No probabilities Murf has no notion of probability. Therefore, we do not model “propagation” of attack probabilities through the finite state system. We also ignore that the intruder may learn some probabilistic information about the participants’ keys by observing the multiple runs of the protocol.
3. No partial information Keys, nonces, etc. are treated as atomic entities. Intruder cannot break such data into separate bits. It cannot perform an attack that results in the partial recovery of a secret.

Modeling IKE

In this paper we have modeled only the main mode protocols. In main mode there are six messages. In the first pair of

messages, "A" sends a cookie and requested cryptographic algorithms, and "B" responds with his cookie and the cryptographic algorithms he will agree to. Messages 3 and 4 are a Diffie-Hellman exchange. Messages 5 and 6 are encrypted with the Diffie-Hellman value agreed upon in messages 3 and 4. In messages 5 and 6, each side reveals its identity and proves it knows the relevant secret.

In general it is assumed that encryption is opaque, however intruder can store the encrypted messages for replaying them later. Signatures are taken to be unforgeable and there exists a Certification Authority which is trusted by both the communicators. Another thing to note here is that we have not modeled the first phase, which includes mutual agreement on the protocols and algorithms

Notation

- * (..) denotes the comments
- * p, g are publicly known
- * a, b are secret random values chosen by A and B resp.

Public signature key main mode:

Protocol A

To begin with, we start with the most basic Diffie-Hellman Key exchange. This step requires the communicators to send over just the Diffie-Hellman values with the identity of the sender.

1. A -> B : Crypto offered
 2. B -> A : Crypto selected
 3. A -> B : $g^a \text{ mod } p$, A
 4. B -> A : $g^b \text{ mod } p$, B
- $K = g^{ab}$ is the session key

Attack on A

There can be a man in the middle attack possible on protocol A. The intruder just changes the Diffie-Hellman exponent but the source of the message remains the same. The responder reacts to this exponent by sending own Diffie-Hellman exponent. The intruder intercepts this message and again replaces the Diffie-Hellman exponent of responder by its exponent. Hence the intruder generates two different session keys with initiator and responder where initiator and responder think that the key they hold is with the intended party.

Protocol B

To prevent the man in the middle attack, we have now added two more steps to the existing protocol A. Here both A and B after exchanging the Diffie-Hellman Values also exchange encrypted signed messages containing all the previous messages sent. Now without knowing the value K , intruder cannot generate correct messages for steps 5 and 6. Thus preventing man in the middle attack.

1. A -> B : Crypto offered
 2. B -> A : Crypto selected
 3. A -> B : $g^a \text{ mod } p$, A
 4. B -> A : $g^b \text{ mod } p$, B
 5. A -> B : $K\{\text{sign on previous messages by A, A}\}$
 6. B -> A : $K\{\text{sign on previous messages by B, B}\}$
- $K = g^{ab} \text{ mod } p$ is the session key

Attack on B

Replay attack is possible on protocol B but here the man in the middle attack is prevented. Man in the middle attack is prevented as the intruder cannot sign messages. When the initiator or responder signs all the previous messages the intruder cannot modify it and the two parties come to know about the Man in the Middle attack by comparing the message they received previously and the messages signed. Replay attack is now possible as the intruder can save the messages and can replay them later. There is no nonce hence replay attack is successful.

Protocol C

To prevent the replay attack, one can add an ephemeral value (nonce) to one of the messages so that the encrypted signed messages would change each time. So replay attack would not be possible.

```
1. A -> B : Crypto offered
2. B -> A : Crypto selected
3. A -> B :  $g_a \text{ mod } p$  , A
4. B -> A :  $g_b \text{ mod } p$  , B , nonce(B)
5. A -> B : K{sign of previous messages by A , A}
6. B -> A : K{sign of previous messages by B, B}
 $K=f(g_a b \text{ mod } p, \text{nonce}(B))$  is the session key
```

Attack on C

Here the replay attack is prevented due to the nonce sent by B. But murf pointed out that the intruder can still impersonate B through replay attack as there is no nonce sent by A.

Protocol D (final)

To prevent the attack found in the previous protocol, we have added one more ephemeral quantity. This time A would generate it. Now intruder can not replay any previous messages since both A and B would issue different nonce when initiating a fresh communication.

```
1. A -> B : Crypto offered
2. B -> A : Crypto selected
3. A -> B :  $g_a \text{ mod } p$  , A, nonce(A)
4. B -> A :  $g_b \text{ mod } p$  , B , nonce(B)
5. A -> B : K{sign of previous messages by A , A}
6. B -> A : K{sign of previous messages by B, B}
 $K=f(g_a b \text{ mod } p, \text{nonce}(B), \text{nonce}(A))$  is the session key
```

Attack on D

Here A sends a nonce to prevent the replay attack on the previous protocol C. Till now there is no attack found on protocol D.

IKE using public encryption key (main mode)**Protocol A**

Basic step for both Public signature and encryption mode can be the same, since in starting protocol neither signatures nor public encryption are used. We start with Diffie-Hellman Key exchange. This step requires the communicators to send over just the Diffie-Hellman values with the identity of the sender.

```
1. A -> B : Crypto offered
```

```

2. B -> A : Crypto selected
3. A -> B :  $g^a \text{ mod } p$  , A
4. B -> A :  $g^b \text{ mod } p$  , B
K= $g^{ab} \text{ mod } p$  is the session key

```

Attack on A

There can be a man in the middle attack possible on protocol A. The intruder just changes the Diffie-Hellman exponent but the source of the message remains the same. The responder reacts to this exponent by sending own Diffie-Hellman exponent. The intruder intercepts this message and again replaces the Diffie-Hellman exponent of responder by its exponent. Hence the intruder generates two different session keys with initiator and responder where initiator and responder think that the key they hold is with the intended party.

Unknown Key Share attack is also possible as the intruder just modifies the source field of the message. The responder thinks the session key is shared between him and the intruder. The intruder replays the unmodified message sent by the responder to the initiator so that the initiator thinks it shares the session key with the responder. Now when an encrypted message comes from the initiator the responder thinks it came from the intruder.

Protocol B

To prevent man in the middle attack, we have a different strategy than what we used in Public signature mode. Here both the parties send an ephemeral value (nonce) encrypted by other party's public key. This alone does not solve the problem. Intruder can also generate encrypted nonce. Therefore the session key is made in combination with the nonce send by the initiator. Now intruder make successfully complete the protocol but will not be able to communicate successfully.

```

1. A -> B : Crypto offered
2. B -> A : Crypto selected
3. A -> B :  $g^a \text{ mod } p$  , { A }B, { nonce(A) }B
4. B -> A :  $g^b \text{ mod } p$  , { B }A , { nonce(B) }A
K=f( $g^{ab} \text{ mod } p$ , nonce(A)) is the session key

```

Attack on B

Here the man in the middle attack is prevented because of the nonce used to generate the key. But replay attack and Unknown Key Share attack both are possible on the above protocol B. Intruder may save all the communication happening between the legitimate parties. And later replay them imposing himself as the initiating party. Even in an on going communication, intruder can capture all the messages send by one party and forward them to the other party, making it believe that intruder has send the messages.

Protocol C (final)

In the previous protocol, both replay and Unknown Key share attack were working since only one of the nonces was used to generate the session key and after transfer of Diffie-Hellman values no proof was asked whether the parties were still the same with whom communication was started. In an attempt to remove replay attack now nonces of both the parties would be used to generate the key. And for removing Unknown Key share attack, two more steps have been added to the protocol. These steps are used to prove that communication is going on between the parties which started it.

```

1. A -> B : Crypto offered
2. B -> A : Crypto selected
3. A -> B :  $g^a \text{ mod } p$  , { A }B, { nonce(A) }B
4. B -> A :  $g^b \text{ mod } p$  , { B }A, { nonce(B) }A
5. A -> B : K{proof I'm A}
6. B -> A : K{proof I'm B}

```

$K=f(gab \bmod p, \text{nonce}(A), \text{nonce}(B))$ is the session key

Attack on C

Replay attack is not possible here due to the 5th and 6th messages. UKS is also not possible as the intruder can not generate any of the proofs. We could not find any other attack possible on this protocol, using Murf. Moreover this protocol C is similar to the corresponding mode in IKE.

Pre-shared Key Main Mode

* A and B have a pre-shared secret J in protocols to follow

Protocol A

A and B share a common secret J. So they can communicate with J as their session key.

Attack on A

The intruder would be able to decrypt the previous conversations if he gets the preshared secret. Hence this protocol doesn't guarantee perfect forward secrecy.

Protocol B

Perfect Forward Secrecy can be obtained if we use different session keys on session. That means we need to create some other value using session key, which could be used as session key. For this purpose we can again use Diffie-Hellman key exchange protocol. And use the key thus obtained in combination with Shared secret to create a session key. Now the session key would be different for each key. Thus Perfect Forward Secrecy is achieved.

1. A -> B : Crypto offered
 2. B -> A : Crypto selected
 3. A -> B : $ga \bmod p, A$
 4. B -> A : $gb \bmod p, B$
- $K=f(gab \bmod p, J)$ is the session key

Attack on B

Replay attack is possible on the above protocol B. Unknown Key Share attack is not possible due to the secret J that only the initiator and responder know. Hence the key generated will be different as responder will use the shared secret with the intruder whereas the initiator will use the shared secret with the responder hence two different keys.

Protocol C

We can use an ephemeral value (nonce) in conjunction with the existing protocol to prevent the replay attack.

1. A -> B : Crypto offered
 2. B -> A : Crypto selected
 3. A -> B : $ga \bmod p, A, \text{nonce}(A)$
 4. B -> A : $gb \bmod p, B, \text{nonce}(B)$
- $K=f(gab \bmod p, J, \text{nonce}(A))$ is the session key

Attack on C

Replay attack is still not prevented in the above protocol C. As we have used nonce only from one party. Unknown Key Share attack is also possible.

Protocol D (final)

To prevent both Replay and Unknown Key share attack. We added two more steps and one more nonce in the generation of session key. This addition leads us to the final protocol.

```

1. A -> B : Crypto offered
2. B -> A : Crypto selected
3. A -> B : ga mod p , A , nonce(A)
4. B -> A : gb mod p , B , nonce(B)
5. A -> B : K{proof I'm A, "A"}
6. B -> A : K{proof I'm B, "B"}
K=f(gab mod p, J, nonce (A), nonce (B)) is the session key

```

Attack on D

Our Murf modeling did not found any other kind of attack possible.

Public Encryption Key, main mode, revised Protocol and encryption of Diffie- hellman values

This protocol functionally is similar to the final protocol reached in Public Encryption Key method, but it is more economic in the sense of computation. This protocol avoids heavy public key encryptions and private key decryption being needed in step 3 and 4 of the original. For this saving it requires generation of secret key using the nonces and all further encryption is done using that secret key.

```

1. A -> B : Crypto offered
2. B -> A : Crypto selected
3. A -> B : KA{ga mod p} , KA{"A"} , KA{nonce(A)} , KA{Alice's cert}
4. B -> A : KB {gb mod p} , KB {"B"} , KB {nonce(B)}
5. A -> B : K{proof I'm A}
6. B -> A : K{proof I'm B}
KA= hash(nonce(A), cookie(A))
KB= hash(nonce(B), cookie(B))
K = f(gab mod p, nonce (A), nonce (B), cookie(A), cookie(B))
cookie(A) and cookie(B) are exchanged in step 1 and 2 respectively

```

When we modeled this revised version, we made a small but significant change while modeling it. We ignored the encryption $KA\{ga \text{ mod } p\}$ and $KB\{gb \text{ mod } p\}$, still we found that it was not susceptible to any attack. Now we again modeled it, exactly as it is. Comparing both we found that there is absolutely no difference in both the models considering the security aspect. On the basis of this result we claim that this secret key encryption can be dispensed with, without affecting the security. This reduction would further lower down the computational requirements of the protocol.

Conclusions

Our study of IKE using formal methods is not a new technique; it has already been done by the using NRL [9], though here we have done it using Murf. However we have tried to bring out the need of the various steps of IKE. This technique of adding more functionality or steps when faced by certain attacks can be used to improve and build protocols.

Here we would like to re-iterate our analysis which points towards the unnecessary encryption of Diffie-Hellman values in the revised Main mode protocol of the Public Key Encryption.

References

1. Charlie Kaufman (edt.): Internet Key Exchange (IKEv2) Protocol. draft-ietfipsec-ikev2-17.txt
2. D. Maughan, M. Schertler, M. Schneider, and J. Turner: Internet security association and key management protocol (ISAKMP), version 1 draft-ietfipsec-isakmp-10.txt (1998).
3. H. Orman: The Oakley key determination protocol, version 2. draft-ietfipsec-oakley-02.txt (1996).
4. D. L. Dill and A. J. Drexler and A. J. Hu and C. H. Yang: Protocol verification as a hardware design aid
5. D. L. Dill: The Murf verification system. In Computer Aided Verification. 8th International Conference, pages 390-3 (1996)
6. Charlie Kaufman, Radia Perlman, Mike Speciner: Network Security. Pearson Education . (2004) pages 441-460
7. John C. Mitchell Mark Mitchell Ulrich Stern: Automated analysis of cryptographic protocols using Murf
8. User Manual, tutorial and other examples provided with the free distribution of Murf.
9. Catherine Meadows: Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer, IEEE, (1999)
10. John C Mitchell, Vitaly Shmatikov, Ulrich Stern: Finite state analysis of SSL 3.0, 7th USENIX Security Symposium, pages 201-15, (1998)
11. Vitaly Shmatikov and U. Stern: Efficient Finite-State Analysis for Large Security Protocols. 11th IEEE Computer Security Foundations Workshop, pages 106-15, (1998)