Refining Computationally Sound Mechanized Proofs for Kerberos

Bruno Blanchet Aaron D. Jaggard Jesse Rao
Andre Scedrov Joe-Kai Tsay

07 October 2009 Protocol eXchange Meeting

Partially supported by ANR, DGA, NSF, OSD/AFOSR, and ONR

Project Background

- Formalization and analysis of Kerberos 5, with and without PKINIT in "public-key mode," using CryptoVerif
  - CryptoVerif [Blanchet] works directly in the computational model
  - Initial work on Kerberos was first computationally sound mechanized proof of an industrial-sized protocol
- Part of an ongoing analysis of the Kerberos 5 suite
  - Previously discovered a flaw in a draft version of PKINIT used in Windows (XP/2000) and Windows Server 2003 [CJSTW]
- PKINIT in particular is complex; Kerberos and PKINIT are available for all major operating systems

Overview of Results

- Proved authentication, secrecy, and usability
  - Keys might be distinguishable from random, but still fine to use [DDMW]
- Define INT-CTXT usability
- Refine earlier CryptoVerif formalization
- Study crypto assumptions needed

Related Protocol Work

- Symbolic analysis of Kerberos (basic and public key) using MSR (includes the attack on public-key mode in PKINIT draft version) [BCJSW'02,'03,'06],[CJSTW'06]
- Computationally sound, by-hand proofs using the BPW framework [BCJST'06]
- By-hand, symbolic correctness proof of IEEE 802.11i and TLS using PCL [HSDDM'05]
- By-hand correctness proof of Diffie-Hellman mode of PKINIT using Computational PCL [RDDM'07]
- ProVerif analysis of PKINIT in DH mode [KT, CSF'09]
- Symbolic analysis of IETF IKE with NRL Protocol Analyzer [M'99]
- Symbolic analysis of Kerberos 4 and TLS using Isabelle [BP'97],[P'97]

(Selected) Mechanized Prover Background

- CryptoVerif: computationally sound, mechanized prover [B'06,'07]
- Beginnings of automation of BPW using Isabelle [BBPSW'06]
- AVISPA tool for automated symbolic validation of protocols and applications [ABBCCCHDHKMMORSTVV'05]
- ProVerif: automatic Dolev–Yao verification tool [B'04]
- Scyther: automatic Dolev–Yao verification tool [C'06]
- Computationally sound, automated symbolic analysis using Casrul tool [CW'05]
- ...

Kerberos

- Goals
  - Repeatedly authenticate a client to multiple servers based on single login
    - Remote login, file access, print spooler, email, directory, ...
- A real-world protocol
  - Part of Windows, Linux, Unix, MacOS, ...
  - Cable TV boxes, high-availability server systems, ...
  - Standardization and ongoing extension/refinement by IETF
  - RFC 4120 (Kerberos 5), RFC 4556 (PKINIT), ...

Abstract Kerberos Messages

$C \longrightarrow K\colon C, T, n_1$
$C \longleftarrow K\colon C, TGT, \{AK, n_1, t_K, T\}_{k_C}$

$C \longrightarrow T\colon TGT, \{C, t_C\}_{AK}, S, n_2$
$C \longleftarrow T\colon C, ST, \{SK, n_3, t_T, S\}_{AK}$

$C \longrightarrow S\colon ST, \{C, t'_C\}_{SK}$
$C \longleftarrow S\colon \{t'_C\}_{SK}$

$(TGT = \{AK, t_K, C\}_{k_T})$
$(ST = \{SK, t_T, C\}_{k_S})$

Abstract Kerberos Messages

$C \longrightarrow K\colon C, T, n_1$
$C \longleftarrow K\colon C, TGT, \{AK, n_1, t_K, T\}_{k_C}$

$C \longrightarrow T\colon TGT, \{C, t_C\}_{AK}, S, n_2$
$C \longleftarrow T\colon C, ST, \{SK, n_3, t_T, S\}_{AK}$

$C \longrightarrow S\colon ST, \{C, t'_C\}_{SK}$
$C \longleftarrow S\colon \{t'_C\}_{SK}$

$(TGT = \{AK, t_K, C\}_{k_T})$
$(ST = \{SK, t_T, C\}_{k_S})$

Abstract Kerberos Messages

$C \longrightarrow K$: C, T, n_1
$C \longleftarrow K$: $C, TGT, \{AK, n_1, t_K, T\}_{k_C}$

$C \longrightarrow T$: $TGT, \{C, t_C\}_{AK}, S, n_2$
$C \longleftarrow T$: $C, ST, \{SK, n_3, t_T, S\}_{AK}$

$C \longrightarrow S$: $ST, \{C, t'_C\}_{SK}$
$C \longleftarrow S$: $\{t'_C\}_{SK}$

$(TGT = \{AK, t_K, C\}_{k_T})$
$(ST = \{SK, t_T, C\}_{k_S})$

Public-Key Kerberos

- Extend basic Kerberos 5 to use public keys
  - Change first round to avoid long-term shared keys (k_C):

$$C \longrightarrow K\colon \mathbf{Cert_C}, [\mathbf{t''_C}, \mathbf{n_2}]_{\mathbf{sk_C}}, C, T, n_1$$
$$C \longleftarrow K\colon \{\{\mathbf{Cert_K}, [\mathbf{k}, \mathbf{ck}]_{\mathbf{sk_K}}\}\}_{\mathbf{pk_C}}, C, TGT, \{AK, n_1, t_K, T\}_{\mathbf{k}}$$

(*ck* is "checksum" over request to which *K* is responding.)

- Motivation/benefits
  - Administrative convenience: Avoid the need to register shared key to use Kerberized services
  - Security: Avoid use of password-derived keys
    - Smartcard authentication support

Public-Key Kerberos

- Extend basic Kerberos 5 to use public keys
  - Change first round to avoid long-term shared keys (k_C):

    $C \longrightarrow K$: $\mathbf{Cert_C}, [\mathbf{t''_C}, \mathbf{n_2}]_{\mathbf{sk_C}}, C, T, n_1$
    $C \longleftarrow K$: $\{\{\mathbf{Cert_K}, [\mathbf{k}, \mathbf{ck}]_{\mathbf{sk_K}}\}\}_{\mathbf{pk_C}}, C, TGT, \{AK, n_1, t_K, T\}_{\mathbf{k}}$

    (*ck* is "checksum" over request to which *K* is responding.)

- Motivation/benefits
  - Administrative convenience: Avoid the need to register shared key to use Kerberized services
  - Security: Avoid use of password-derived keys
    - Smartcard authentication support

Cryptographic Assumptions

- Public-key encryption assumed to satisfy IND-CCA2
- Signature scheme assumed to satisfy UF-CMA
- Symmetric encryption assumed to satisfy IND-CPA and INT-CTXT (and thus IND-CCA2 and INT-PTXT [BN'00])
- Hash function assumed to be collision resistant
  (We'll return to this.)

Cryptographic Assumptions

- Public-key encryption assumed to satisfy IND-CCA2
- Signature scheme assumed to satisfy UF-CMA
- Symmetric encryption assumed to satisfy IND-CPA and INT-CTXT (and thus IND-CCA2 and INT-PTXT [BN'00])
- Hash function assumed to be collision resistant (We'll return to this.)

Authentication I

Using CryptoVerif, we can show that the following hold with overwhelming probability for basic and public-key Kerberos:

- Authentication (injective) of the KAS to the client
  - If an honest client receives what appears to be a valid reply from the KAS, then the KAS generated a reply for the client.
- Authentication of request for service ticket
  - If an honest TGS processes a valid request for a service ticket, then the ticket in the request was generated by the KAS and the authenticator included in the request was generated by the client.

Authentication II

Again, with overwhelming probability for basic and public-key Kerberos:

- Authentication (injective) of TGS to client
- Authentication of request to server
- Authentication of server to client

Key Secrecy

- In both basic and public-key Kerberos, we have:
  - Secrecy of *AK*
    - If an honest client *C* finishes an AS exchange with the KAS that generate the authentication key *AK* for use between *C* and an honest TGS, then *AK* is secret w.r.t. the *real-or-random* definition of secrecy.
  - Secrecy of *SK*
    - If an honest client finishes a TG exchange with an honest TGS that generated the service key *SK* for use between *C* and an honest server *S*, then *SK* is secret with respect to the *real-or-random* definition of secrecy.
- These keys will be distinguishable from random once they are used for encryption in the subsequent requests.

Subsession Key Secrecy

- The final round of Kerberos can be used by C and S to agree on a subsession key for further use
  - This key can be generated by either the client or the server
- CryptoVerif proves that both basic and public-key Kerberos preserve:
  - Secrecy of the key possessed by the party that generated the subsession key
  - One-session secrecy of the key possessed by the other party of $\{C, S\}$
  - Difference from possibility of replays
    - Party accepting (not generating) key might accept same key multiple times, allowing it to be distinguished from random
    - Current formalization lacks replay cache

Key Usability

- Notion of *key usability* introduced by Datta, Derek, Mitchell, and Warinschi [2006]
- Weaker than being indistinguishable from random
  - Captures whether a key is still 'good' for future cryptographic operations
- Important for protocols that perform operations with a key during a run and allow future use of the key
- Definition parallels definition of key indistinguishability
  - Two-phase attacker (Ae, Ac): Ae interacts with protocol session, then Ac tries to win an attack game that uses the exchanged keys, *e.g.*, IND-CCA2 against an encryption scheme
  - During the second phase, Ac cannot interact with protocol sessions

Key Usability with CryptoVerif

Stronger version of key usability (w.r.t. IND-CCA2 encryption)—adversary can still interact with uncompleted protocol sessions during the attack game:

- Adversary A first interacts with polynomially many protocol sessions
- A requests a session ID to be drawn at random; let k be the key locally output in that session
- A is given access to LR-encryption oracle E_k and a decryption oracle D_k corresponding to k
- A plays a variant of the IND-CCA2 game where:
  - A may interact with uncompleted protocol sessions,
  - But all sessions do not accept ciphertexts output by E_k when they reach a point of the protocol at which at least one session expects to receive a message encrypted under k

Key Usability in Kerberos

We can use CryptoVerif to prove

- Usability of *AK*:
  - If an honest client *C* finishes a session of basic or public-key Kerberos involving the KAS and an honest TGS, then the authentication key *AK* is (strongly) usable for IND-CCA2-secure encryption.
- Usability of *SK*:
  - If an honest client *C* finishes a session of basic or public-key Kerberos involving the KAS, an honest TGS, and an honest server, then the service *SK* is (strongly) usable for IND-CCA2-secure encryption.

Weakening Crypto

- Leak contents of authenticators
  - $\{C, t\}_{AK}, C, t$ instead of just $\{C, t\}_{AK}$
  - $\{C, t'\}_{SK}, C, t'$ instead of just $\{C, t'\}_{SK}$
  - This was suggested by examining by-hand proofs in Dolev–Yao model
- The authentication results still hold for both basic and public-key Kerberos
- The secrecy of the subsession key also still holds for both basic and public-key Kerberos
- Advantage of CryptoVerif—very fine control over cryptographic assumptions

(Strong) INT-CTXT Usability

Not previously defined; how to define?

- In standard INT-CTXT game, attacker tries to produce a valid ciphertext that was not generated by the encryption oracle
- In the key-usability game, the attacker interacts with protocol sessions
  - If these include encryptions by protocol participants, the attacker can trivially win the game
  - This possibility highlighted by the use of CryptoVerif
- Currently check (strong) INT-CTXT usability of a session key k by having the decryption oracle refuse:
  - Ciphertexts produced by the encryption oracle
  - Ciphertexts produced by participants (using *any* session key)

Game for INT-CTXT Usability

- First, $\mathcal{A}$ is given the security parameter η and $\mathcal{A}$ can interact, as an active adversary, with polynomially many protocol sessions of Σ.
- At some point, at the request of $\mathcal{A}$, a session identifier sid is drawn at random and $\mathcal{A}$ is given access to a encryption oracle $\mathcal{E}_k(.)$ and a decryption oracle $\mathcal{D}_k(.)$, both keyed with a key k locally output in session sid.

- Adversary $\mathcal{A}$ plays a variant of an INT-CTXT game in which:
  - $\mathcal{A}$ may submit messages m to $\mathcal{E}_k(.)$, which returns $\mathcal{E}_k(m)$;
  - $\mathcal{A}$ never queries $\mathcal{D}_k(.)$ on a cyphertext output by $\mathcal{E}_k(.)$;
  - $\mathcal{A}$ may interact with uncompleted protocol sessions; and
  - if k is a key that is used at least once for encryption, then $\mathcal{A}$ never queries $\mathcal{D}_k(.)$ on a cyphertext encrypted by any key playing the role of k in any one of the protocol sessions. (*I.e.*, in Kerberos, if k is an authentication, resp. service, key that is used at least once for encryption, then $\mathcal{A}$ never queries $\mathcal{D}_k(.)$ on a cyphertext encrypted by *any* authentication, resp. service, key.)
- The experiment outputs 1 (and $\mathcal{A}$ wins) if the decryption oracle properly decrypts a query by the adversary, *i.e.*, outputs $m \neq \perp$, otherwise the experiment outputs 0.

(Strong) INT-CTXT Usability

- CryptoVerif can handle this definition
  - Is it the right one? (It seems reasonable.)
- Proofs of INT-CTXT usability of *AK* and *SK*
  - Recall that we have secrecy/one-session secrecy for the subsession key

The Problem

- Pointed out by Chao Feng (National University of Defense Technology, Hunan, China)
- In earlier formalization, only honest clients could get certificates
- This precluded insider attacks by dishonest clients

The Solution

- Added oracle to allow the attacker to obtain certificates for dishonest clients
  - Without this, CryptoVerif proves security for broken version of PKINIT
  - However, without these oracles we couldn't prove security using earlier version of CryptoVerif
- Added oracle to allow the attacker to obtain certificates for dishonest KASes
  - KAS is assumed to be honest, so this doesn't have practical effect
- We can prove all the same results as before

PKINIT Message Flow

$C \longrightarrow K$: $\mathbf{Cert_C}, [\mathbf{t''_C}, \mathbf{n_2}]_{\mathbf{sk_C}}, C, T, n_1$

$C \longleftarrow K$: $\{\{\mathbf{Cert_K}, [\mathbf{k}, \mathbf{ck}]_{\mathbf{sk_K}}\}\}_{\mathbf{pk_C}}, C, TGT, \{AK, n_1, t_K, T\}_{\mathbf{k}}$

- *ck* is "checksum" over request to which *K* is responding
- This is a keyed hash; various options in specification

The Checksum Question

In checking the implementation of these oracles, the following issue arose:

- K sends C the message

$$\{\{Cert_K, [k, ck]_{sk_K}\}\}_{pk_C}, C, TGT, \{AK, n_1, t_K, T\}_k$$

- ck is a checksum, keyed with k, over the request received by K
  - This is intended to bind K's response to C's request
  - Added after MITM attack discovered against draft version of PKINIT [CJSTW]
- If acting as a client, the adversary receives the k used to compute ck; might this help an attack?

Effects on Automated Proof

- If K replies to the adversary (as a client), the adversary then gets the key k used to compute ck in that response
- One scenario:
  - Adversary sends PKINIT request to K (as a client) and receives a response with k
  - C sends PKINIT request to which the adversary responds with k
    - In CryptoVerif, C gets name of TGS from the adversary
    - If TGS namespace is large, the adversary may be able to influence the C's request so that the checksum (computed using k) matches that previously signed by K
    - Adversary could then reply to C, replaying checksum signed by K
- In general, revelation of k to adversary seems to cause problems with proofs

Observations I

- If we assume *ck* is the output of a collision-resistant hash function, CryptoVerif is able to prove security as described
- If we assume *ck* is a UF-CMA MAC, then CryptoVerif cannot complete the proof
  - Attacker knows the key *k* used to compute *ck* and can influence the message over which *ck* is computed
  - The adversary learning the key

Observations II

- Temporal arguments are outside the scope of CryptoVerif. However:
  - k is freshly generated by K and the adversary only learns k *after* ck has been computed
  - After obtaining $[k, ck]_{sk_K}$ and learning k, the adversary must induce C to send a request whose checksum (computed with k) is ck
  - The only part of C's message that the adversary can influence is the name of the TGS
  - If the set of possible TGSes isn't too large, the adversary won't have enough flexibility
- We believe this is a modeling challenge, not a viable attack

Summary

- Proof of authentication and secrecy properties of PKINIT using the tool CryptoVerif
- Survey of initial results
  - Extended our Kerberos analysis project to include mechanized proofs
  - First mechanized proof of authentication and secrecy for a commercial/real-world protocol directly in the computational model
  - Stronger notion of IND-CCA2 usability suitable for CryptoVerif
- Definition of (strong) INT-CTXT usability
  - Suitable for use with CryptoVerif
  - Proofs for Kerberos keys
- Added oracles needed for insider attacks
- Study of crypto assumptions for checksums

Future Work

- Refine INT-CTXT key usability definitions?
- Using weaker crypto (both for checksums and generally)
  - What properties are required (or implicitly assumed)?
  - What properties do we need?
- Add more details from Kerberos RFCs
  - Adding lots of detail to key generation may cause games to explode