Formal Analysis of Kerberos 5

A. Scedrov
University of Pennsylvania

Partially supported by ONR and NSF

Cervesato, Jaggard, Scedrov, Tsay, Walstad

Breaking and Fixing Public-Key Kerberos

- Part of ongoing formal analysis of Kerberos 5 suite
  - Previously studied core part of protocol and cross-realm authentication
  - Focus on PKINIT, public-key extension to Kerberos
- Attack on PKINIT found when using “public-key mode” (one of two possible modes)
  - Breaks binding client’s request and the response
  - Prevents full authentication and confidentiality
- Formal verification of fixes preventing attack
  - Close, ongoing interactions with IETF Working Group
- Our work caused an August 2005 Microsoft security patch for Windows 2000, Windows XP, and Windows 2003

  www.microsoft.com/technet/security/bulletin/MS05-042.mspx

PKINIT Attack and Fixes (Overview)

- Protocol level attack on PKINIT-25
  - Not a problem with crypto or implementation
  - Kerberos server believes he is talking to the attacker
  - Client believes she is talking to the Kerberos server
  - Attacker knows the key shared by the client and Kerberos server
- Possible because the Kerberos server does not sign data identifying the client
  - Attacker constructs request based on client's request
  - Kerberos server signs data from client, sends in reply to attacker
  - Attacker forwards this to client after learning keys
  - Ran Canetti, consulted on details of spec., independently hypothesized the possibility of an "identity misbinding" attack
- PKINIT-27 intended to defend against this attack
  - Kerberos server signs data derived from client's identity

Impact

- Our work caused August 2005 Microsoft security patch and is cited there
  www.microsoft.com/technet/security/bulletin/MS05-042.mspx
- Vulnerability in linux, in the Heimdal protocol (linux version of Kerberos)
- Although other vulnerabilities viewed as more pressing for IT managers, this attack has real-world effects and highlights a design vulnerability
  - Remote code execution, privilege elevation seem to arise from coding errors, not design flaws
  - No known exploit using our attack

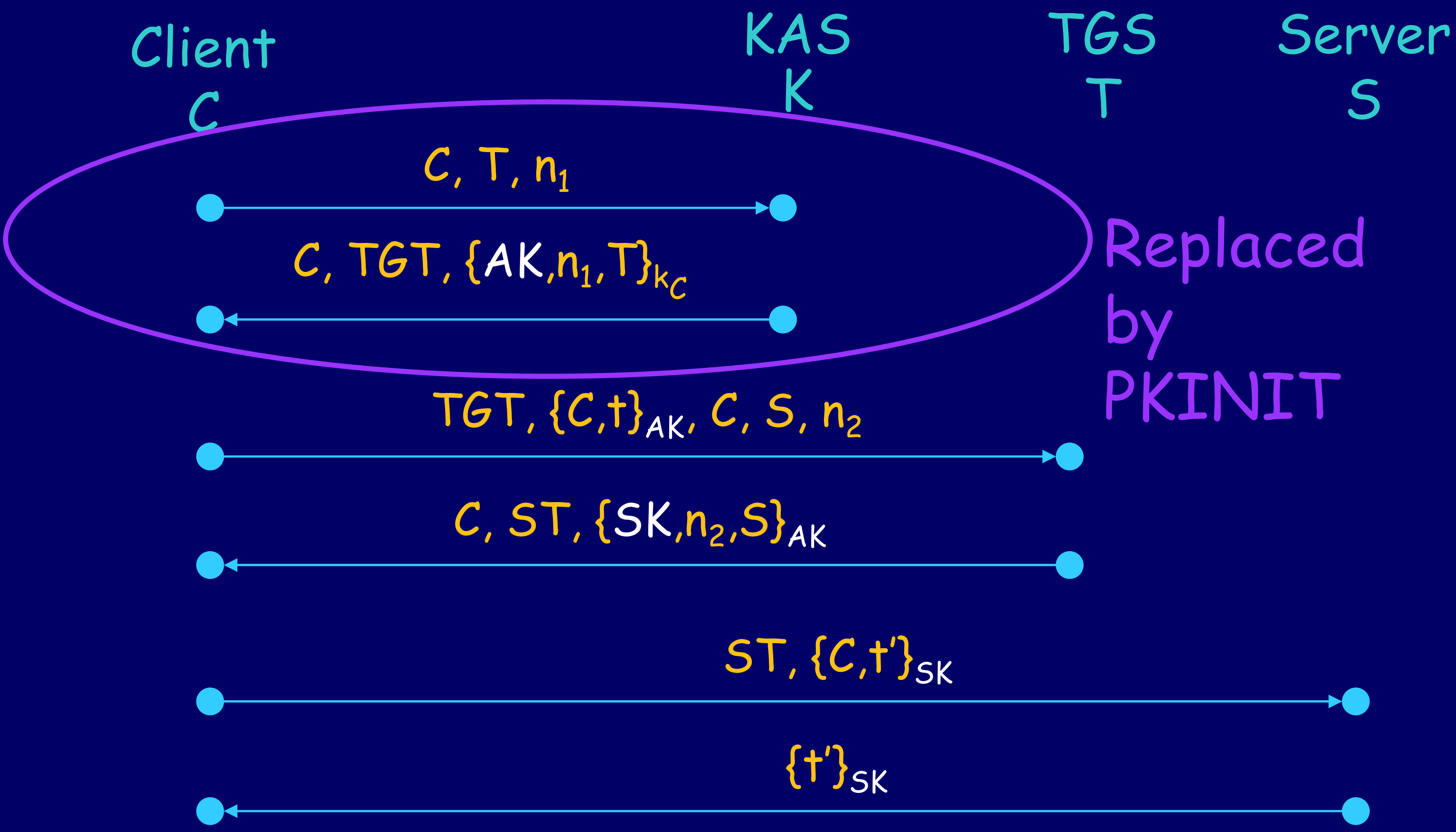
Interactions with CERT and IETF

- Providing forensics to CERT
- Close collaboration with IETF Kerberos WG
  - Discussed possible fixes we were considering
  - Attack announced on WG list in July 2005
  - We verified a fix the WG suggested
    - This was incorporated into PKINIT-27 and into RFC 4556
  - Presented this work at IETF-63
    - Discussed possible fixes and our analysis of these
    - Useful discussions with WG participants on other areas for work
  - Participating in WG subsequent meetings
- Impact of formal methods in IETF security area
  - At security-area level, they want to see more interaction with formal methods

Kerberos Overview

- Goals
  - Repeatedly authenticate a client to multiple servers on single log-on
    - Remote login, file access, print spooler, email, directory, …
- A real world protocol
  - Part of Windows, Linux, Unix, Mac OS, …
  - Cable TV boxes, high availability server systems, …
  - Standardization and ongoing extension/refinement by IETF (very active --- 10 documents)
    - Current version for Kerberos is RFC 4120

Abstract Kerberos Messages



Client
C
KAS
K
TGS
T
Server
S
C, T, n_1
$C, TGT, \{AK, n_1, T\}_{k_C}$
$TGT, \{C, t\}_{AK}, C, S, n_2$
$C, ST, \{SK, n_2, S\}_{AK}$
$ST, \{C, t'\}_{SK}$
$\{t'\}_{SK}$
Replaced by PKINIT

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

Public-Key Kerberos

- Extend basic Kerberos 5 to use PKI
  - Change first round to avoid long-term shared keys
  - Originally motivated by security
    - If Kerberos server is compromised, don't need to regenerate shared keys
    - Avoid use of password-derived keys
  - Current emphasis on administrative convenience
    - Avoid the need to register in advance of using Kerberized services
- This extension is called PKINIT
  - Current version is RFC 4556
  - We found attack in -25; -26 does not change the relevant design
  - Versions included in Windows and Linux (called Heimdal)
  - Implementation developed by CableLabs (for cable boxes)
  - Apparently not in MIT version

Two Modes

- In general, no key k_C shared between C and K
  - Credentials for C instead encrypted under a temporary key k
    - How to generate and deliver k?
- Public-key encryption
  - k is generated by K
  - k encrypted under C's public key and is signed by K
  - Attack is against this mode
- Diffie-Hellman
  - k is generated by DH using data from C and K
  - C and K each send signed data to contribute to DH key
    - Option for 'reuse' of the shared secret
  - CableLabs appears to be only implementation of this
  - Initial inspection did not turn up attacks against this mode

Public-Key Encryption Mode

$$C \xrightarrow{t_C,\ n_2,\ Cert_C,\ [t_C,\ n_2]_{skC},\ C,\ T,\ n_1} K$$

$$C \xleftarrow{\{\{Cert_K,\ k,\ n_2,\ [k,\ n_2]_{skK}\}\}_{pkC},\ C,\ \{k_{CT}, C\}_{k_T},\ \{k_{CT}, n_1, T\}_k} K$$

$$C \xrightarrow{\{k_{CT},\ C\}_{k_T},\ \{C\}_{k_{CT}},\ C,\ S,\ n_3} T$$

$$C \xleftarrow{C,\ \{k_{CS},\ C\}_{k_S},\ \{k_{CS},\ n_3,\ S\}_{k_{CT}}} T$$

$$C \xrightarrow{\{k_{CS},\ C\}_{k_S},\ \{C,\ t\}_{k_{CS}}} S$$

$$C \xleftarrow{\{t\}_{k_{CS}}} S$$

Formalizing the Request

- Our formalization of pa-data includes
  - t_C = cusec/ctime (in pkAuthenticator)
  - n_2 = nonce (in pkAuthenticator)
  - $[t_C, n_2]_{skC}$ = signature (in signerInfos) over t_C, n_2 using C's secret key skC
- Our formalization of req-body includes
  - C = cname
  - T = sname
  - n_1 = nonce

$$t_C, n_2, [t_C, n_2]_{skC}, C, T, n_1$$

Formalizing the Reply

- Our formalization of pa-data includes
  - k = replyKey (in ReplyKeyPack)
  - n_2 = nonce (in ReplyKeyPack), from AS-REQ
  - $[k, n_2]_{skK}$ = signature with K's secret key skK
  - $\{\ldots\}_{pkC}$ is encryption with C's public key pkC
- C = cname in AS-REP
- X = ticket in AS-REP
- Our formalization of enc-part includes
  - AK = key
  - n_1 = nonce
  - t_K = authtime
  - T = sname
  - $\{\ldots\}_k$ is encryption with the reply key k

$$\{k, n_2, [k, n_2]_{skK}\}_{pkC}, C, X, \{AK, n_1, t_K, T\}_k$$

PKINIT Attack and Fixes (Overview)

- Protocol level attack on PKINIT-25
  - Not a problem with crypto or implementation
  - Kerberos server believes he is talking to the attacker
  - Client believes she is talking to the Kerberos server
  - Attacker knows the key shared by the client and Kerberos server
- Possible because the Kerberos server does not sign data identifying the client
  - Attacker constructs request based on client's request
  - Kerberos server signs data from client, sends in reply to attacker
  - Attacker forwards this to client after learning keys
  - Ran Canetti, consulted on details of spec., independently hypothesized the possibility of an "identity misbinding" attack
- PKINIT-27 intended to defend against this attack
  - Kerberos server signs data derived from client's identity

The Attack

At time t_C, client C requests a ticket for ticket server T (using nonces n_1 and n_2):

$$C \xrightarrow{t_C,\ n_2,\ Cert_C,\ [t_C,\ n_2]_{skC},\ C,\ T,\ n_1} I$$

The attacker I intercepts this, puts her name/signature in place of C's:

$$I \xrightarrow{t_C,\ n_2,\ Cert_I,\ [t_C,\ n_2]_{skI},\ I,\ T,\ n_1} K$$

Kerberos server K replies with credentials for I, including: fresh keys k and AK, a ticket-granting ticket X, and K's signature over k,n_2:

(Ignore most of enc-part)

$$I \xleftarrow{\{k,\ n_2,\ [k,\ n_2]_{skK}\}_{pkI},\ I,\ X,\ \{AK,\ \ldots\}_k} K$$

I decrypts, re-encrypts with C's public key, and replaces her name with C's:

$$C \xleftarrow{\{k,\ n_2,\ [k,\ n_2]_{skK}\}_{pkC},\ C,\ X,\ \{AK,\ \ldots\}_k} I$$

- **I knows fresh keys k and AK**
- C receives K's signature over k,n_2 and assumes k, AK, etc., were generated for C (not I)

- Principal P has secret key skP, public key pkP
- $\{msg\}_{key}$ is encryption of msg with key
- $[msg]_{key}$ is signature over msg with key

Consequences of the Attack

- The attacker knows the keys C uses; she may:
  - Impersonate servers (in later rounds) to the client C
  - Monitor C's communications with the end server
- Other notes
  - Attacker must be a legal user
  - C is authenticated to end server as attacker (not as C)
  - DH mode appears to avoid this attack
    - Still need to formally prove security for DH

After the First Round

- Both the attacker I and client C know the keys k and AK
  - C believes the KDC produced k and AK for C
- Attacker may monitor communications
  - Attacker must put her name into the TGS-REQ and AP-REQ messages to match the tickets
  - Attacker learns keys in TGS-REP and AP-REP
- Attacker may impersonate servers
  - Instead of forwarding modified –REQ messages, attacker may simply forge –REP messages herself

Desired Authentication Property

If a client C processes a message containing KDC-generated public-key credentials, then some KAS K produced a set of such credentials **for C**.

- The attack shows this property does not hold in pk-init-25/-26
- This property holds if:
  - The KAS signs k, F(C, n_i); or
  - The AS-REP is as in pk-init-27

Preventing the Attack in General

- Sign data identifying client
  - The KDC signs k, $F(C, n_i)$
  - Assume $F(C, n) = F(C', n')$ implies $C = C'$ and $n = n'$
  - AS-REQ message now formalized as

    $$\{k, F(C, n_i), [k, F(C, n_i)]_{skK}\}_{pkC}, C, X, \{AK, n_1, t_K, T\}_k$$

- We have a formal proof that this guarantees authentication
  - Does cname/crealm uniquely identify client?
  - Added secrecy properties if $F(C, n)$ identifies pkC?

pk-init-27 and the Attack

- In the change implemented in pk-init-27:
  - The KDC signs k, cksum (*i.e.*, cksum in place of n_2)
    - k is replyKey
    - cksum is checksum over AS-REQ
    - Easier to implement than signing C, k, n_2
  - AS-REP now formalized as

$$\{k, cksum, [k, cksum]_{skK}\}_{pkC}, C, X, \{AK, n_1, t_K, T\}_k$$

- We have a formal proof that this guarantees authentication
  - Assume checksum is preimage resistant
  - Assume KDC's signature keys are secret
  - Plan to carry out a more detailed, cryptographic proof in the future

ReplyKeyPack in pk-init-26

```
ReplyKeyPack ::= SEQUENCE {
   replyKey        [0] EncryptionKey,
     -- Contains the session key used to encrypt the
     -- enc-part field in the AS-REP.
   nonce           [1] INTEGER (0..4294967295),
     -- Contains the nonce in the PKAuthenticator of the
     -- request.
... }
```

ReplyKeyPack in pk-init-27

```
ReplyKeyPack ::= SEQUENCE {
   replyKey            [0] EncryptionKey,
     -- Contains the session key used to encrypt the
     -- enc-part field in the AS-REP.
   asChecksum          [1] Checksum,
     -- Contains the checksum of the AS-REQ
     -- corresponding to the containing AS-REP.
     -- The checksum is performed over the type AS-REQ.
     -- The protocol key [RFC3961] of the checksum is the
     -- replyKey and the key usage number is 6.
     -- If the replyKey's enctype is "newer" [RFC4120]
     -- [RFC4121], the checksum is the required
     -- checksum operation [RFC3961] for that enctype.
     -- The client MUST verify this checksum upon receipt
     -- of the AS-REP.
... }
```

Corrected Public-Key Kerberos

$$\begin{tikzcd}
\text{Client } C \arrow[rrr, "{\mathrm{Cert}C,\ [t_C,\ n_2]_{skC},\ C, T,\ n_1}"] & & & \text{KAS } K \\
C & & & K \arrow[lll, "{\{\{\mathrm{Cert}_K,\ [k,ck]_{skK}\}\}_{pkC},\ C,\ TGT,\ \{AK,n_1,T\}_k}"']
\end{tikzcd}$$

$$\boxed{TGT = \{AK,C,t_K\}_{k_T}\ ,\ ck = \mathrm{Hash}_k(\mathrm{Cert}C,\ [t_C,\ n_2]_{skC},\ C,T,\ n_1)}$$

- Extend basic Kerberos 5 to use Public Keys
  - Change first round to avoid long-term shared keys (k_c)

- Motivations
  - Administrative convenience: Avoid the need to register in advance of using Kerberized services
  - Security: Avoid use of password-derived keys
    - Smartcard authentication support instead

Cryptographically Sound Proofs of Security Properties of Kerberos

- Proofs by hand use the Cryptographic Library by Backes, Pfitzmann, and Waidner
  - Pair of system models: An abstract ideal cryptographic library and a real cryptographic library
  - Ideal cryptographic library is a Dolev-Yao-style deterministic formalism
  - Results in the ideal cryptographic library hold for the real cryptographic library (real system "as secure as" ideal system)
  - This requires implementation of provably secure crypto primitives
    - E.g. IND-CCA2 asymmetric encryption, UF-CMA signature, IND-CCA2 + INT-CTXT symmetric encryption

Backes, Cervesato, Jaggard, Scedrov, Tsay

Cryptographically Sound Proofs of Security Properties of Kerberos

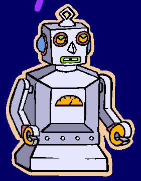
- Our cryptographic results for Kerberos and for Kerberos with corrected PKINIT:
  - Server-Client Entity Authentication: If the server completes a protocol run, apparently with client C, then, with overwhelming probability, C started the protocol with some KAS K and requested a service ticket from some TGS. Moreover, if a client C completes a protocol run, apparently with server S, then, with overwhelming probability, S sent a valid reply (=last protocol message) to C.
  - Key Secrecy: An optional subsession key exchanged between server and client is indistinguishable from a fresh random key for any polynomial time adversary

Mechanized proofs of security

- Current work: Blanchet, Jaggard, Tsay, Scedrov
- Cryptographically sound proof of authentication
- Blanchet's tool CryptoVerif based on polynomial-time probabilistic process calculus [Lincoln, Mitchell, Ramanathan, Scedrov, Teague]
- Subtleties with crypto assumptions

Mechanization Context

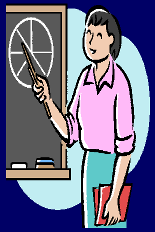
Analysis of Cryptographic Protocols



Mechanized Proofs

Symbolic/
Dolev-Yao

Academic
Protocols



e.g.
- NSL
- Otway-Rees
- Yahalom

- Algebra of terms
- Good for checking protocol structure
- Limited adversary capabilities

Using strong Crypto

Computational

Commercial
Protocols

Kerberos/PKINIT

e.g.
- TLS
- Kerberos
- IKE

- Complexity theory
- Probability theory
- Strong security guarantees

Hand proofs in Computational model prone to human error, and
even in Dolev-Yao model highly time consuming for more complex protocols

Mechanization Overview (1)

- Formalization and Analysis of Kerberos 5 with and without its public-key extension PKINIT (in Public-Key mode), a public-key extension to Kerberos 5, using the CryptoVerif tool
- First computationally sound mechanized proof of a full industrial-sized protocol
  - Especially PKINIT is complex, involving both asymmetric and symmetric cryptographic primitives
  - Kerberos and PKINIT are available for all major operating systems, e.g. implemented in Microsoft Windows (Vista/XP/2000) and Windows Server 2003
- Generalization of Key Usability notion

Mechanization Overview (2)

- Part of an ongoing analysis of Kerberos 5 suite
  - Previously discovered a flaw in a draft version of PKINIT used in Windows (XP/2000) and Windows Server 2003
    - Joint work with Cervesato and Walstad
  - Previously conducted by-hand computational proofs of PKINIT and Kerberos
    - Joint work with Cervesato and Backes using the *Backes-Pfitzmann-Waidner model (BPW)*
- CryptoVerif tool works directly in the computational model
  - So far tested only on *academic* protocols, e.g. NSL, Otway-Rees, Yahalom
  - Our work provides evidence for the suitability of CryptoVerif for *industrial* protocols

Related Protocol Work

- [Butler, Cervesato,Jaggard, Scedrov,Walstad '02, '03, '06], [Cervesato,Jaggard,Scedrov,Tsay,Walstad '06]: Symbolic analysis of Kerberos (basic and public-key) using Multi Set Rewriting (Includes the attack on PKINIT draft version)
- [Backes,Cervesato,Jaggard,Scedrov,Tsay '06]: Computational Sound by-hand Proofs of Kerberos using the BPW model
- [He,Sundararajan,Datta,Derek,Mitchell '05]: By-hand symbolic correctness proof of IEEE 802.11i and TLS using Protocol Composition Logic
- [Roy,Datta,Derek,Mitchell '07]: By-hand correctness proofs of Kerberos (incl. Diffie-Hellman mode of PKINIT) using Computational Protocol Composition Logic
- [Meadows '99] : Symbolic analysis of IETF IKE with NRL protocol analyzer
- [Bella,Paulson '97] / [Paulson '97]: Symbolic analysis with Isabelle theorem prover of Kerberos 4 / TLS

…

More Mechanized Prover Background

- [Blanchet'06,'07], [Blanchet,Pointcheval '06]: CryptoVerif; computationally sound mechanized prover
- [Backes,Basin,Pfitzmann,Sprenger,Waidner '06]: Beginnings of automation of BPW using Isabelle theorem prover
- [Armando,Basin,Boichut,Chevalier,Compagna,Cuellar,Hankes Drielsma,Heám,Kouchnarenko,Mantovani,Mödersheim, von Oheimb,Rusinowitch,Santiago,Turuani,Viganò,Vigneron '05]: AVISPA tool for automated symbolic validation of protocols and applications
- [Blanchet '04]: ProVerif; automatic Dolev-Yao verification tool
- [Cremers '06]: Scyther; automatic Dolev-Yao verification tool
- [Cortier,Warinschi '05]: Computationally sound, automated symbolic analysis using Casrul tool

…

Cryptographic Assumptions

- Public-key encryption assumed to be IND-CCA2, signature scheme assumed to be UF-CMA
- Symmetric encryption implemented as *encrypt-then-MAC*, with IND-CPA encryption and (W)UF-CMA message authentication code
  - This implies IND-CCA2 and INT-PTXT [Bellare,Namprempre'00]
- Hash function is collision resistant

Authentication Properties (1)

- We can show with CryptoVerif that following holds with overwhelming probability

1. Authentication of the KAS to the client [inj]
   - If an honest client receives what appears to be a valid reply from the KAS, then the KAS generated a reply or the client
2. Authentication of request for ST
   - If an honest TGS processes a valid request for a service ticket ST, then the ticket in the request was generated by the KAS and the authenticator included in the request was generated by the honest client (modulo the MACs).
3. Authentication of TGS to client [inj]
   - If an honest client sees that appears to be a valid reply to a request for a ST for an honest server S from an honest TGS, then the TGS generated a reply for the client.

Authentication Properties (2)

4. Authentication of request to server
   - If an honest server S processes a valid request, ostensibly from an honest client C, containing a service ticket ST and a session key pair (SK, mSK), then some honest TGS generated (SK, mSK) for C to use with S and also created ST (modulo the MAC). Furthermore, C created the authenticator (modulo the MAC).

5. Authentication of server to client
   - If an honest client C sees a valid reply from an honest server S, then this reply was generated by S (modulo the MAC).

Key Secrecy Properties

1. Secrecy AK
   - If an honest client C finishes an AS exchange with the KAS, where the KAS generated the authentication key pair (AK, mAK) for the use between C and an honest TGS T, then AK and mAK are secret w.r.t. the *real-or-random* definition of secrecy
2. Secrecy of SK
   - If an honest client finishes a TG exchange with an honest TGS, where the TGS generated the service key pair (SK, mSK) for the use between C and an honest server S, then SK and mSK are secret with respect to the *real-or-random* definition of secrecy

- Note: The keys AK and SK will no longer be indistinguishable from random once they are used in a client C's request to the TGS T and the server S, respectively

Key Usability

- Notion of *Key Usability* introduced by Datta, Derek, Mitchell, and Warinschi in 2006
- Weaker than key indistinguishability
- Important for protocols that perform operations with a key during a run and allow for the future use of this key
- An exchanged key is *usable* if it is `good' for future cryptographic operations
  - Definition parallels definition of key indistinguishability
  - Two phase attacker (A_e, A_c): first A_e interacts with protocol sessions, then A_c tries to win an attack game that uses exchanged key, e.g. IND-CCA2 against an encryption scheme
  - During second phase, A_c cannot interact with protocol sessions

Key Usability with CryptoVerif

- Stronger version of key usability (w.r.t to IND-CCA2 encryption), where adversary can still interact with uncompleted protocol sessions during the attack game:
  - The adversary A first interacts with polynomial many protocol sessions
  - At the request of A, a session id *sid* is drawn at random and A is given access to LR-encryption oracle E_k and a decryption oracle D_k , where k is the key locally output in *sid*
  - A plays variant of an IND-CCA2 game where
    - A may interact with uncompleted protocol sessions
    - But all sessions of the protocol 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 the key k

- Discussion:
  - Stronger notion (at the very least)
  - More realistic ?
  - Yet another definition of key usability (+ Comp Thm) ?

Key Usability in Kerberos

1. 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 pair (AK, mAK) is (strongly) usable for IND-CCA2 secure encryption (under mentioned crypto assumptions)

2. 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 S, then the session key pair (SK, mSK) is (strongly) usable for IND-CCA2 secure encryption (under mentioned crypto assumptions)

Conclusions (1)

- Extended formalization of Kerberos 5
  - Cross-realm and public-key cases
- Found a MITM attack against public-key encryption mode in PKINIT-25 / PKINIT-26
  - Protocol attack with industrial impact (MS security bulletin)
  - Formulated a general fix defending against this attack
- Close collaboration with IETF WG
  - Discussion and analysis of possible fixes
    - We've analyzed the fix employed in PKINIT-27
- Cryptographically sound security proofs of security properties of basic Kerberos 5 and of corrected public-key Kerberos, by hand and in the CryptoVerif tool

Conclusions (2)

- Proof of authentication and secrecy properties of basic and public-key Kerberos using the tool CryptoVerif
  - Extended our Kerberos analysis project to include mechanized proofs
- First mechanized proof of authentication and secrecy for a full commercial/real-life protocol directly in the computational model
  - CryptoVerif seems suitable for industrial protocols
- Stronger version of key usability
  - Proved mechanically for Kerberos

Future work

- Using weaker crypto
- Stay closer to Specs
  - Adding additional fields from specs
- Yet another notion of Key Usability ?
- Diffie-Hellman mode of PKINIT
  - Mechanized proof in the computational model
    - Hand Proof exists in Computational PCL [Roy,Datta,Derek,Mitchell '07]
- Other protocols: web services, SOA, privacy, trust, …

Thank You!

Formal Analysis of Kerberos 5

A. Scedrov
University of Pennsylvania

Partially supported by ONR and NSF

Proof Sketch for General Defense

- Assume
  - Client receives AS-REP with $[k, F(C, n_i)]_{skK}$
  - KAS's signature key is secret
  - Signatures are unforgeable
  - $F(C, n) = F(C', n')$ implies $C = C'$ and $n = n'$
- Proof sketch
  - Signature in reply must come from the KAS K
  - K would only produce this signature in response to a request containing C' such that $F(C', n') = F(C, n)$
  - Collision-freeness of F implies that K created the reply for C

Proof Sketch for pk-init-27

- Assume
  - Client receives AS-REP as in pk-init-27
  - KAS's signature key is secret
  - Signatures are unforgeable
  - Checksums are collision-free
- Proof sketch
  - Signature in AS-REP must come from the KAS K
  - K would only produce this signature in response to an AS-REQ whose checksum is the signed value
  - Collision-freeness of checksums implies that the AS-REQ was as claimed (including C's name)

CryptoVerif Basics (1)

- CryptoVerif (CV) can prove secrecy properties and correspondence asssertions for cryptographic protocols, and also cryptographic primitives
  - Secrecy w.r.t. real-or-random definition
  - Authentication through [injective] correspondence assertions [inj:] φ ==> [inj:] ψ
  - Proof of cryptographic primitives in the random oracle model
- CV works directly in the Computational Model
  - Protocols represented as processes in calculus inspired by pi-calculus, the calculi by [Lincoln,Mitchell,Ramanathan,Scedrov,Teague '98, '99, '02] and [Laud '05]; with probabilistic semantics
  - Processes Q and Q' are *observationally equivalent* ($Q \approx Q'$) if, intuitively, an adversary has negligible probability of distinguishing Q from Q'

CryptoVerif Basics (2)

- Proofs as sequences of games
  - Construct sequence $Q_0 \approx Q_1 \approx \ldots \approx Q_{n-1} \approx Q_n$, where Q_0 formalizes the investigated protocol and desired security properties are obvious in Q_n
  - CV uses cryptographic and syntactic transformations to reach Q_j from Q_{j-1}
- Subtleties with crypto assumptions
- Note: CryptoVerif is sound but not complete
  - Properties it cannot prove are not necessarily invalid
  - CV operates in different modes:
    - Automatic mode (if only symmetric crypto is used)
    - Interactive mode (if public-key crypto is used)
      - Requires user to type in commands that determine the next game transformation
- Static corruption of protocol participants

CryptoVerif Basics (3)

Little example:

$Q_C = !^{i_C <= N}\ c_2[i_C]\ (h_T : tgs);$ new $n_1 : nonce;$

$\overline{c_3[i_C\ ]}\langle C, h_T, n_1\rangle;$

$c_4[i_C\ ]\ (= C, m_1 : maxmac, mac_1 : macs, m_2 : maxmac, mac_2 : macs);$

if $check(m_2\ , mK_C\ , mac_2\ )$ then

let $injbot(concat1(AK\ , mAK\ , = n_1\ , = h_T\ )) = dec(m_2, K_C\ )$ in

event $e_C(h_T, n_1, m, m_2)$ …

CryptoVerif proves authentication of K to C by proving the query:

inj-event($e_C(T\ , n, x, y)$) $\Rightarrow$ inj-event($e_K(C, T\ , n, z\ , y)$)

- Runtime: Authentication properties of
  - Basic Kerberos: ca. 7 s, 70 game transformations
  - Public-key Kerberos: ca. 1 min 40 s, 124 game transformations

Definition: Strong Key Usability

Let Π=(K, E, D) $\in$ S a symmetric encryption scheme, b $\in$ {0,1}, Σ a key exchange protocol, an adversary. Consider following experiment

Exp$^{b}_{A, \Sigma, \Pi}(\eta)$:

- First, A is given η and A can interact with polynomially many sessions of Σ
- At some point, at the request of A, a session identifier *sid* is drawn at random and A is given access to a LR-encryption oracle E_k(LR(.,.,b)) and an decryption oracle D_k(.), where k is locally output in *sid.*
- At some point A plays a variant of an IND-CCA2 attack game
  - Where A submits same-length pairs to E_k(LR(.,.,b)), never queries D_k(.) on outputs by E_k(LR(.,.,b))
  - A may still interact with uncompleted protocol sessions, but all sessions of the protocol do not accept ciphertexts output by E_k(LR(.,.,b)) when they reach a point in the protocol in which at least one session expects to receive a message encrypted under the key k.
- At some point A outputs a guess bit d, which is also the output of **Exp**$^{b}_{A, \Sigma, \Pi}(\eta)$
- Define the advantage of adversary A by **ADV**$^{ke}_{A, \Sigma, \Pi}(\eta)$ = |Pr(**Exp**$^{1}_{A, \Sigma, \Pi}(\eta)$ =1) - Pr(**Exp**$^{0}_{A, \Sigma, \Pi}(\eta)$ =1)|.
- Then key k is *strongly usable* (for IND-CCA2 encryption) for schemes in S if for all $\Pi \in$ S and all ppt A, **ADV**$^{ke}_{A, \Sigma, \Pi}(\eta)$ is negligible.