Key management... Dude, wait, what?

Eric Rescorla
RTFM, Inc.
ekr@rtfm.com

IETF 71

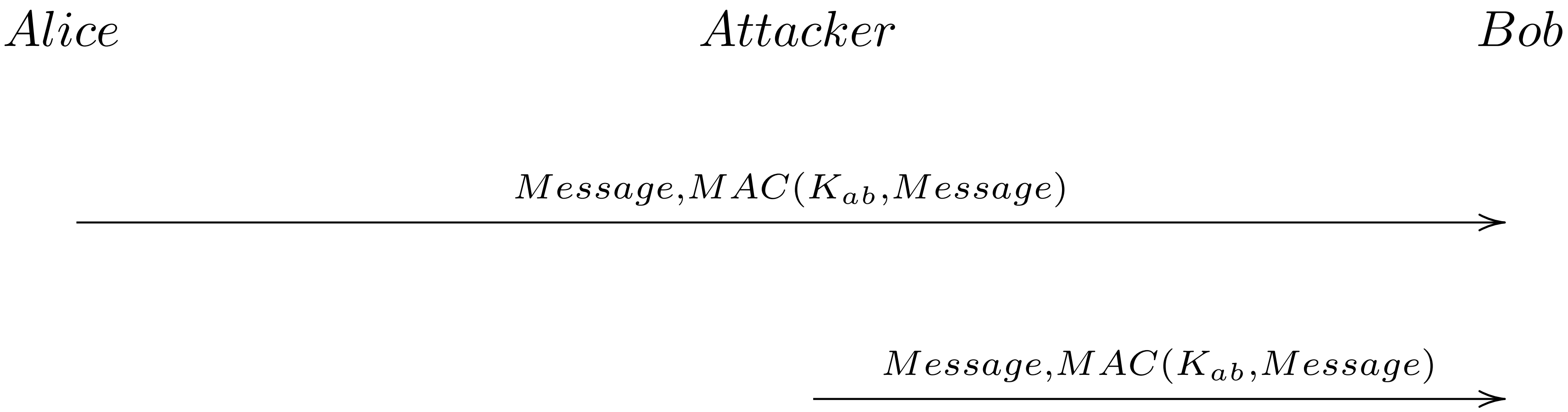
Current State of Routing Protocol Security[*]

- Focus is on integrity and data origin authentication
  - This data is not confidential
- Routing protocols use manual key management
  - Shared keys are configured at the router interface
    - This usage model is critical to preserve
  - Traffic is protected with ad-hoc MACs based on those keys
- Why is this bad?
  - Replay and cut-and-paste attacks (see David Ward's presentation)
  - The MACs are generally pretty weak—and hard to upgrade

[*] We're just talking about securing adjacencies here

What’s a replay attack?

- Alice and Bob share a key (K_{ab})
  - But never change it



Alice *Attacker* *Bob*

$Message, MAC(K_{ab}, Message)$

$Message, MAC(K_{ab}, Message)$

- Requires an on-path attacker
- Works whenever association parameters are repeated
  - Keys
  - Other meta-data protected by the MAC (e.g., host/port)

Defenses against replay attack

- Fresh association parameters
  - Implicit
    - Example: TCP
      - Each connection has its own host/port quartet
      - If in the MAC then you can't replay between connections
      - ... unless you get a host/port collision
  - Explicit
    - Establish a fresh connection identifier
      - Force it to be unique
- Fresh keys for the association
  - Using a key management protocol
  - This is the standard COMSEC solution
    - Provides generic security without trusting the main protocol

What's a key management protocol?

- We have a number of elements $\mathbf{E} = E_1, E_2, ...$ that want to communicate
  - They have some long term credentials
    - $*$ Shared symmetric key/password
    - $*$ Asymmetric key pairs
    - $*$ Keys $+$ certificates
  - Generally can't use these keys directly for communication
- A key management protocol establishes shared cryptographic state
  - Traffic protection keys
  - Algorithms and other parameters

Why use a KMP?

- Security
  - Establish fresh keys
    - ∗ Prevents replay and cut-and-paste attacks
    - ∗ Good “cryptographic hygiene”
  - Explicit liveness and peer authentication
- Performance
  - Public key cryptography is slow
  - Faster to establish symmetric keys and then use them
- Capability discovery
  - Get peer’s certificate
    - ∗ Not an issue in systems with manual configuration
  - Discover/negotiate algorithms
  - Allows uncoordinated capability upgrade

Deployment Scenarios

- Unicast
  - This is a technically well understood problem (TLS, IPsec, ...)
    - Issue is mapping it onto actual protocols with minimal disruption
  - BGP, LDP
    - Run over TCP
    - Not the topic of this meeting (TCP-AO, etc.)
  - IS-IS, OSPF, RIP
- Multicast/broadcast
  - Less technically well understood
    - Some experience in MSEC WG (GDOI, GSAKMP)
  - IS-IS, OSPF, RIP only

A trivial unicast key management protocol

- Assume Alice, Bob share a key: K_{ab}

$$Alice \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad Bob$$

$$\xrightarrow[Algorithms=HMAC-SHA1,HMAC-SHA256]{Name=Alice,Random}$$

$$\xleftarrow[Algorithm=HMAC-SHA1]{Name=Bob,Random}$$

- Two new keys:

$$K_{a \to b} = HMAC(K_{ab}, Random_{Alice} || Random_{Bob})$$

$$K_{b \to a} = HMAC(K_{ba}, Random_{Bob} || Random_{Alice})$$

Similar protocols can be used with asymmetric keys

Key Management for Multicast Groups

- We have a set of elements $\mathbf{E} = E_1, E_2, ...$
  - They have some long term credentials
  - We want them to share a single symmetric key, parameters, etc.
- Classic solution: Have a master element (group controller)
  - Generates group key
  - Forms unicast associations with each element
  - Pushes group key to each group member
    - $*$ Send $E(K_{E_i}, K_{group})$ to node E_i
- Examples: GDOI, GSAKMP

Key Management with a Group Master

$$\begin{tikzcd}
 & Master \arrow[dl, "{E(K_{E_1},K_{group})}"] \arrow[d, "{E(K_{E_2},K_{group})}"'] \arrow[dr, "{E(K_{E_3},K_{group})}"] & \\
E_1 & E_2 & E_3
\end{tikzcd}$$

Integrity with a Group Key

$$\begin{tikzcd}
E_1 \arrow[r, "{M_1,MAC(K_{group},M_1)}"] \arrow[d, "{M_1,MAC(K_{group},M_1)}"'] \arrow[dr, "{M_1,MAC(K_{group},M_1)}"] & E_2 \\
E_3 & E_4
\end{tikzcd}$$

Problems with Group Key Management

- The current schemes need a master/group controller
  - Who runs this?
  - Why do you trust them?
- Impersonation of other group members
  - All group members have the same MAC key
    - ∗ Used for both MAC generation and MAC verification
  - Any group member can impersonate any other group member
- When is this a problem?
  - When group members are mutually suspicious
  - Is this true for these protocols?

Operating without a dedicated master

- Option 1: joint key establishment
  - Everyone works together to establish a group key
  - e.g., group Diffie-Hellman
  - What happens when a new member joins/leaves?
    - Generate a new key
    - Or have some node act as master for it
- Option 2: elect a master
  - What happens if that master leaves?
  - New election...
- Either of these would require significant new protocol work

Dealing with impersonation

- Basic problem is use of one symmetric key
  - Used for both MAC generation and verification
  - Need to separate these functions
- Option 1: Public key cryptography (digital signature)
  - Every message is signed
  - Need to somehow establish all the key pairs
    - PKI? Pairwise agreements? Bootstrap CA protocol?
- Option 2: TESLA
  - Needs a master
    - Which you need to trust
  - We have no operational experience with TESLA
    - The timing is known to be tricky

What problem were we trying to solve again?

- Hey this all looks pretty expensive
- Replay attack
  - This can be fixed by hacking the non-security part
  - Add unique per-association identifiers inside the MAC
- Key agility
  - This can be fixed (clumsily) by adding a key identifier
- Stronger algorithms and algorithm agility
  - This can be fixed (clumsily) by tying algorithms to keys
- Impersonation of other group members
  - This looks pretty hard to fix
  - How bad is it?