FIT5037: Network Security
Security at Application layer

Faculty of Information Technology
Monash University

April 30, 2019

Copyright Notice

Lecture 8: Security at Application layer

Lecture Topics:

- Symmetric key cryptography
- Asymmetric key cryptography
- Pseudorandom Number Generators and hash functions
- Authentication Methods and AAA protocols
- Security at Network layer (IPsec)
- Security at Network layer (firewalls and wireless security)
- Security at Transport layer
- **Security at Application layer**
- Computer system security and malicious code
- Computer system vulnerabilities and penetration testing
- Intrusion detection
- Denial of Service Attacks and Countermeasures / Revision

Outline

- Electronic Mail Security
  - brief review of PGP and S/MIME
  - mention of DKIM
- Secure Shell Protocol
- DNS Service Security

Electronic Mail Security

Security of Electronic Mail Messages

Electronic Mail Security

- The basis for email over the Internet
  - Simple Mail Transfer Protocol (SMTP originally specified in RFC821 August 1982, latest: RFC5321 October 2008)
  - Message syntax (originally specified in RFC822, latest: 5322)
  - Multipurpose Internet Mail Extension (MIME specified in RFC 2045-2049)
- Neither SMTP nor the message syntax supports security services
- Required security properties:
  - confidentiality: protection from disclosure
  - authentication: of sender of message
  - message integrity: protection from modification
  - non-repudiation of origin: protection from denial by sender
- Why providing transport layer security is not enough for SMTP protocol?
- Two main schemes for end-to-end security of email messages:
  - PGP
  - S/MIME

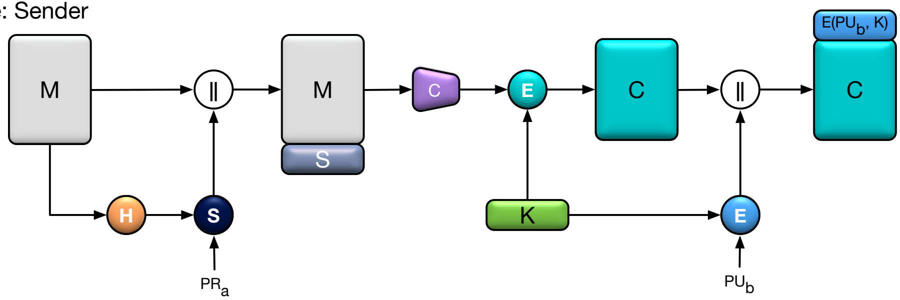
Pretty Good Privacy (PGP)

- PGP was developed by Philip R. Zimmerman
- PGP provides confidentiality and authentication services that can be used for electronic mail and file storage applications
- RFC 4880: OpenPGP Message Format is an Internet standard
  - Referred to as PGP 5.x
  - GnuPG or GPG is an OpenPGP implementation (available as an open source project)
- PGP General Functions
  - key management
    - key ring for public and private key storage
    - per-message session keys
  - authentication and data integrity
    - digital signature
  - compression performed
    - after digital signature
    - before encryption
  - confidentiality
    - encryption
  - Radix-64 (Base64) conversion
    - interoperability of text representation between different mail servers and relay agents

PGP: Authentication and Confidentiality (Sender)

- PGP can provide authentication only or authentication and confidentiality

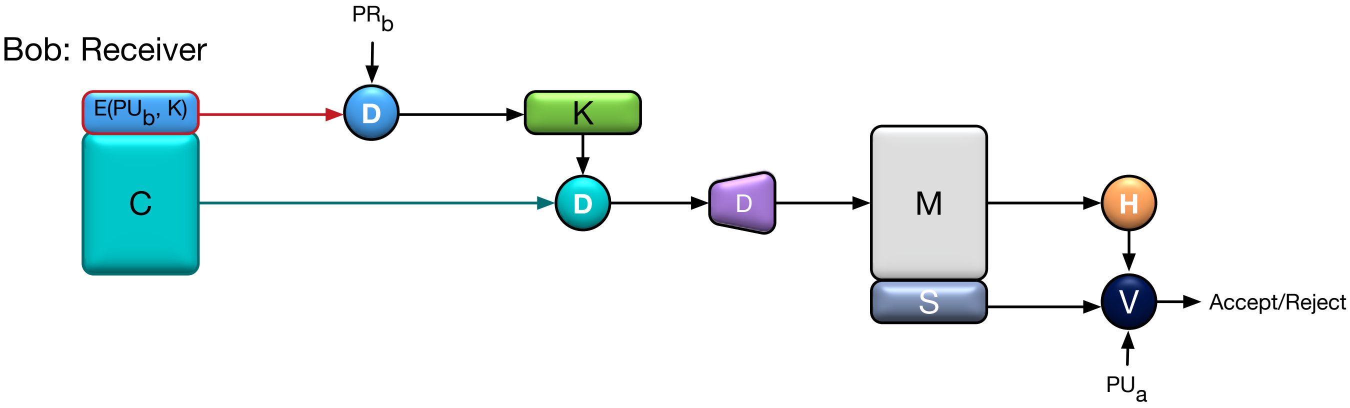
Alice: Sender



Providing both services on the same message:

- Sender first signs the message with its own private key
- Encrypts compressed (message + signature) with a session key
- Encrypts the session key with the recipient's public key
- and the encrypted key precedes the rest

PGP: Authentication and Confidentiality (Receiver)



- Receiver first decrypts the session key using his private key
- Decrypts compressed (message + signature) with the recovered session key
- Verifies the signature with sender's public key
- Stores message + signature if verified

PGP Public Key Management: Web of Trust

- Rather than relying on certificate authorities, in PGP every user is one's own CA
  - can sign keys for users they know directly
- Forms a "web of trust"
  - trusted keys will be signed
  - can trust keys others have signed if there is chain of signatures to them
- Key ring includes trust indicators
- Users can also revoke their keys

Secure Multipurpose Internet Mail Extensions (S/MIME)

- original SMTP protocol can only transmit text messages
- MIME provides support for varying content types and multi-part messages
  - encodes binary data in textual form
  - converts 8-bit binary values to printable characters
- S/MIME[1] added security enhancements to MIME
  - is very similar to PGP
- creates a MIME body according to Cryptographic Message Syntax (CMS)[2]
- Also offers the ability to sign and/or encrypt messages with the following functions
  - **enveloped data**: encrypted content and associated keys
  - **signed data**: encoded (message + signed digest)
  - **clear-signed data**: clear-text message + encoded signed digest
    - recipient with MIME capability but not S/MIME capability can read the message
  - **signed and enveloped data**: nesting of signed and encrypted entities, i.e., various orderings for encrypting and signing

---

1 "S/MIME version 3.2"

2 Cryptographic Message Syntax

S/MIME Certificate Use

- S/MIME uses X.509 v3 certificates
- managed by using a hybrid of a strict X.509 CA hierarchy and PGP's web of trust
- each client has a list of trusted CA's certificates
- and own public/private key pairs and certificates
- certificates must be signed by trusted CA's
- S/MIME agents must provide some certificate retrieval mechanism
  - using X.500 directory service
  - using secure DNS to associate certificates with domain names for S/MIME (RFC8162)

Domain Keys Identified Mail (DKIM)

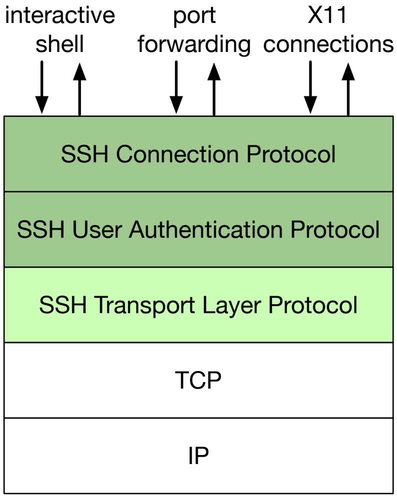
- a specification for cryptographically signing email messages by domains rather than users
- so signing domain claims responsibility
  - ADMD: Administrative Management Domain
  - SDMD: Signing Domain Identifier
- recipients / agents can verify signature
  - DKIM separates the identity of the signer from the author of the message
  - the recipient can interpret the header fields of DKIM signatures and decide how to process the message
- proposed Internet Standard (latest) RFC6376
  - has been widely adopted
  - provides authenticity and message integrity for the header fields as well as the body of the messages
  - relies on Domain Name Services (DNS) for key management
  - does not provide confidentiality

Secure Shell (SSH)

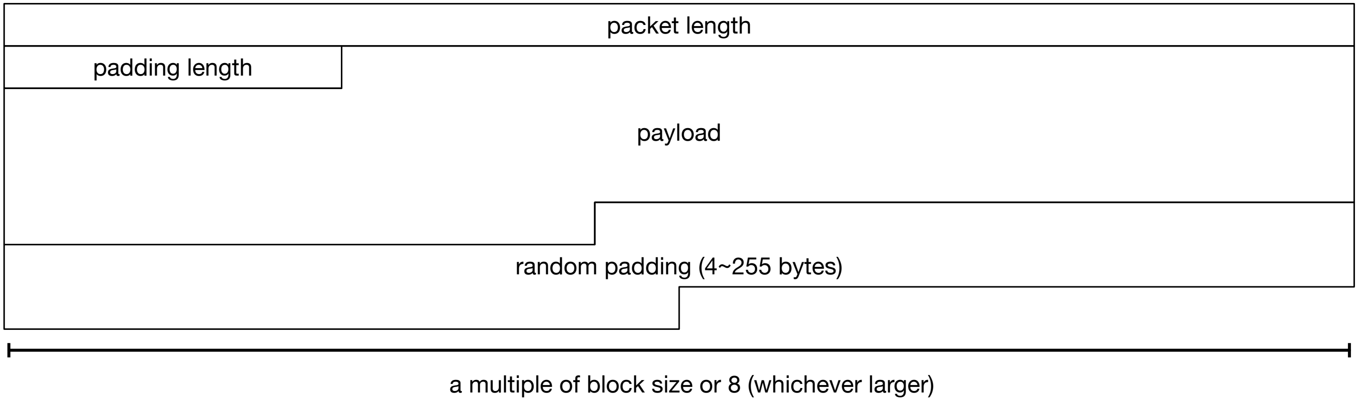
Secure Shell: A secure protocol for remote login

Secure Shell (SSH) Protocol

- protocol architecture is defined in RFC 4251
- runs over a reliable transport layer protocol (such as TCP)
- comprised of three major components:
  - Transport Layer Protocol defined in RFC 4253
    - server authentication
    - confidentiality
    - integrity
    - perfect forward secrecy
    - optional compression
  - (client) Authentication Protocol defined in RFC 4252
    - authenticates the client to the server
  - Connection Protocol defined in RFC 4254
    - multiplexes the encrypted tunnel into several logical channels



SSH Transport Layer Protocol: General Message Structure



SSH Transport Layer Protocol: Confidentiality and Integrity

```
message = packet_length || padding_length || payload || padding
C = E(K, message)
T = MAC(K, sequence_number || message)
```

- `sequence_number`: a 32-bit unsigned integer which is implicit (not included with the message)
  - set to `0` for the first message
    - incremented for every message regardless of encryption or MAC being used
- MAC algorithm is chosen independently at each side
  - recommended to be the same in practice
- MAC is transmitted unencrypted `C || T`
  - this approach is referred to as Encrypt-and-MAC and is vulnerable for some combinations of ciphers and MACs[3]

[3] Bellare, Mihir, Tadayoshi Kohno, and Chanathip Namprempre. "Authenticated encryption in SSH: provably fixing the SSH binary packet protocol." Proceedings of the 9th ACM conference on Computer and communications security. ACM, 2002.

SSH Transport Layer Protocol: Key Exchange

- Kex starts with each side sending list of supported algorithms
  - if the first algorithm is the same in both client and server list it will be selected
    - otherwise server will iterate through client list to find one that server also supports
  - if explicit server authentication is used the key exchange messages are signed

- each side may guess the kex algorithm and send its public key
  - `first_kex_packet_follows` indicates this
  - if the guess is wrong the packet will be ignored
- `kex_algorithms`: DH kex and a hash function
  - initially modulo prime DH groups supported
    - support for ECC public key and ECDH kex are added in RFC 5656
  - group parameters are defined for the protocol (separate from IKE groups)
  - hash function is used to derive keys from shared secret (Appendix)

client and server key exchange message:

```
byte          SSH_MSG_KEXINIT
byte[16]      cookie (random bytes)
name-list     kex_algorithms
name-list     server_host_key_algorithms
name-list     encryption_algorithms_client_to_server
name-list     encryption_algorithms_server_to_client
name-list     mac_algorithms_client_to_server
name-list     mac_algorithms_server_to_client
name-list     compression_algorithms_client_to_server
name-list     compression_algorithms_server_to_client
name-list     languages_client_to_server
name-list     languages_server_to_client
boolean       first_kex_packet_follows
uint32        0 (reserved for future extension)
```

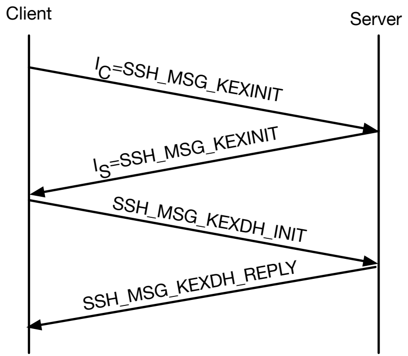
- RFC 6239: Suite B Cryptographic Suites for Secure Shell (SSH)
  uses ECDH, ECDSA, AES-GCM and SHA-256 and 386, and X.509 certificates

SSH Transport Layer Protocol: Public Key Authentication

- supports public key method for server authentication
  - defines key format, signature and encryption algorithm, encoding of signature and encrypted data
    - DSS and RSA raw keys
    - OpenPGP certificates
    - X.509 certificates (RFC 6187)
- client may have received the public key of the server (host key) securely (outside protocol)
  - otherwise the protocol is vulnerable to MitM attack
- server signs its key exchange message with its private key
  - the key exchange also contains the host (public) key
- client verifies the host key
  - this authenticates the shared secret from kex
    - provided that public key of server is authentic

SSH Transport Layer Protocol: Kex and Auth

- In the following exchanges:
  - C: client:
  - S: server
  - V_C: client id
  - V_S: server id
  - PU_S: server's public (host) key
- SSH_MSG_KEXINIT negotiate supported algorithms
  - message structure shown in a previous slide
  - I_C: client's kex init message
  - I_S: server's kex init message
- SSH_MSG_KEXDH_INIT: DH_C
  - DH public key of the client
- SSH_MSG_KEXDH_REPLY: $PU_S || DH_S || sig$
  - $H = hash(V_C || V_S || I_C || I_S || PU_S || DH_C || DH_S || K)$
  - $sig = Sign(PR_S, H)$
  - DH_S: is server's DH public key
  - K is: the DH shared secret
- Encryption and MAC keys are derived from K (Appendix)

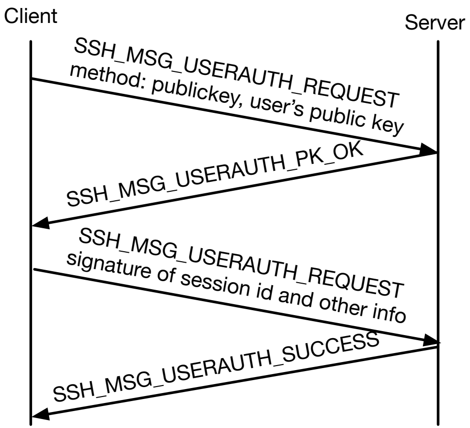


SSH (client) Authentication Protocol

- receives from the lower level:
  - the session identifier
  - whether confidentiality is provided
- client starts with `SSH_MSG_USERAUTH_REQUEST`
  - providing a `user name`
  - `service name` (there may be more than one service provided)
  - `method name`
  - `method` specific fields
- server tells client what authentication methods can be used
  - `publickey`
  - `password`
  - `hostbased`
  - `none`
  - additional methods can be defined
- server responds with
  - `SSH_MSG_USERAUTH_FAILURE` if request is rejected
    - whether authentication can continue providing a list of methods
  - `SSH_MSG_USERAUTH_SUCCESS` is request is accepted

    is sent only after authentication sequence is completed

client `publickey` authentication



SSH Connection Protocol

- runs on top of SSH Transport Layer and (client) Authentication protocols
- multiplexes different types of user communication using channels over the same transport layer connection
  - terminal sessions, forwarded TCP connections, X11 forwarding etc.
- channels are identified by numbers
- each side can open a channel by sending `SSH_MSG_CHANNEL_OPEN`
  - `channel type`: a string specifying the type of channel
  - `sender channel`: local identifier of the channel
  - `initial window size`: how many bytes can be sent without adjusting windows (flow control)
  - `maximum packet size`
  - channel data
- other sides responds with
  - `SSH_MSG_CHANNEL_OPEN_CONFIRMATION`
  - or `SSH_MSG_CHANNEL_OPEN_FAILURE`
- further channel data is sent with `SSH_MSG_CHANNEL_DATA` messages
- when no more date is sent by a party it will send `SSH_MSG_CHANNEL_EOF`
- when either party wishes to terminate a channel it sends `SSH_MSG_CHANNEL_CLOSE`

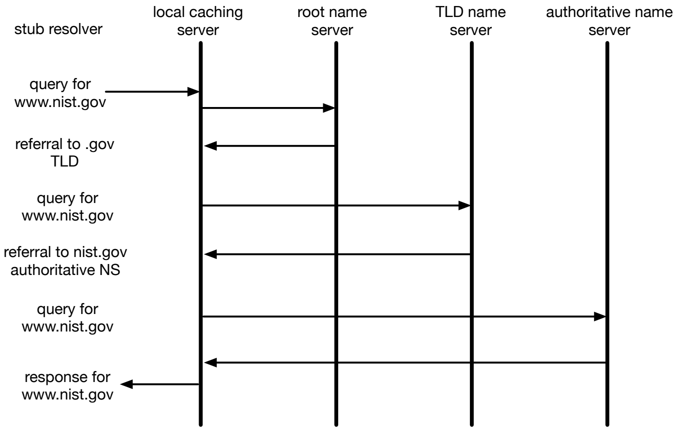
Security of Domain Name Service

Domain Name Service (DNS)

- DNS resolves domain names to network (IP) addresses
  - for network devices IP addresses are unique identifiers
  - to access resources using user-friendly names domain names are used
- DNS infrastructure is comprised of geographically distributed entities
  - uses a hierarchical model
    - *root domain* is represented as a dot ("."): root servers resolve the next level queries
    - *Top Level Domain (TLD)*: TLD servers resolve second-level domains
    - *authoritative* name servers for each domain resolve names under that particular domain
  - DNS cache servers store a copy of resolved names to provide a faster response
- DNS data is intended to be accessible to any device in the Internet
  - confidentiality for public information is not a concern
  - integrity and origin authenticity of DNS information is the main goal for DNS security

DNS Request Resolution Process

- DNS resolution process as described in NIST SP-800-81-2:



- *stub resolver*: a process resolving user queries
- *recursive name server* or *resolving name server*: a caching DNS server
  - will check its cache regarding requested name and responds if found
  - will recursively check the cache or query TLD/root servers until query is resolved
    - will send back an error if not found after all queries

Attacks on DNS

- Exploit DNS host
- Masquerade
  - an adversary spoofs the identity of a DNS node
    - denies access to services identified by the spoofed node service
    - provides bogus information poisoning DNS cache servers
- Violation of integrity of DNS information
  - stored on the authoritative DNS
  - stored on a DNS cache server
  - in transit
- Use DNS as a reflect/amplify Distributed Denial of Service attack
  - DNS uses UDP protocol vulnerable to this type of attack (spoofed source IP)
    - no connection is required
    - response will be sent to spoofed IP

DNS Security

- Secure DNS host
  - secure operating system
    - patch/update OS
    - least privilege etc.
  - secure applications
    - remove unnecessary software
    - patch/update software
  - process isolation
  - network fault tolerance
    - server redundancy
- Protect DNS management transactions
  - outlined as TSIG in RFC 2845 and updated by RFC 3645
  - update of DNS records
  - data replication between DNS nodes
- Protect DNS query/response transactions
  - outlined as DNSSEC in RFC 4033
  - can offload the cryptographic operations from clients to DNS servers
- NIST SP-800-81-2: Secure Domain Name System (DNS) Deployment Guide

DNS Data

- *Zone*: a configurable entity within a name server that contains the domain name information
  - Owner name: domain name
  - TTL: time to live
  - Class: currently one class IN
  - RRType: Resource Record Types defined in RFC 1035 (and updated by several RFCs)
    - `SOA`: marks the start of a zone authority
    - `A`: alias, provides the IP address for a host name
    - `MX`: mail exchanger
    - `NS`: an authoritative name server
    - `CNAME`: a canonical name for an alias
  - RData: Resource information

DNS Server Types

- authoritative: server is an authoritative source for RRs of a particular zone
  - *master* or *primary*: contains (authoritative) zone files created and edited manually by zone administrator or updated dynamically by trusted clients
  - *slave* or *secondary*: also contains (authoritative) zone files created through replication from master server by *zone transfer* DNS transaction
    - added to provide fault-tolerance
    - zone transfer: is a name resolution query with type code `AXFR` meaning "all RRs for the zone"
    - the slave server is notified by *NOTIFY* transaction when a zone file in master server changes
- caching (also resolving/recursive): resolves name queries by asking authoritative name servers (in the hierarchy) or from its built-up cache (from previous queries)
  - generally local name server resolving queries on behalf of local clients
  - a name server can be both authoritative and caching
    - an authoritative DNS server that resolves recursive queries may be vulnerable to compromise of its zone information

DNS Transactions

- DNS query/response
  - a query is made by a stub resolver
  - a response may come from an authoritative or caching server
- Zone Transfer
  - the method by which the secondary server updates its zone files
  - a zone transfer query requests all RRs from the primary server
  - originates from the secondary server
    - either as a response to DNS NOTIFY
    - or based on the value of `Refresh` field in `SOA` section of zone file
  - this transaction reveals much more information than a DNS query
- Dynamic Updates
  - add/delete individual RRs, delete specific set of RRs e.g. same owner name, RRType etc., delete an existing domain (all RRs), add a new domain
  - when enabled open the server to various attacks
- DNS NOTIFY
  - signals the secondary server to initiate a zone transfer
  - more efficient than frequent polling
  - fraudulent DNS NOTIFY messages could trigger unnecessary zone transfer (may require transmission of large amount of information), can be prevented by limiting the sender to primary server

DNS Data Contents: Threats

- Lame Delegation
  - FQDN or IP address of name servers are changed in child zone but not in the parent zone
    - child zone becomes unreachable
- Zone Drift
  - `Refresh` and `Retry` fields of SOA RR of the primary are set to too high values (in-frequent refresh)
    - frequent zone file change may lead to inconsistency between primary and secondary zone files
- Zone Thrash
  - `Refresh` and `Retry` fields of SOA RR of the primary are set to too low values (frequent refresh)
    - frequent zone file change may lead to high workload on both primary and secondary and may lead to DoS
- Information for Targeted Attacks
  - `HINFO` and `TXT` resource records can provide more information for an adversary targeting the organisation
    - e.g. information about software name and version etc.

DNS Data Content: Protective Measures

- analyse DNS RRs data for security implications
- use a zone file integrity checker tool to check for necessary constraints
  - `Refresh`: secondary server to wait between zone transfers
    - e.g. updated frequently: 20m-2h, infrequently: 2h-12h
  - `Retry`: secondary server to wait before attempting zone transfer after failure
    - a fraction of `Refresh` value e.g. 5m-1h
  - `Expire`: length of time to consider zone information valid if primary server is unreachable
    - a multiple of `Refresh` e.g. 2w-4w
  - Minimum `TTL`: the default value of TTL of zone RRsets
    - tells clients how long RRS should be cached
    - depends on how often zone file changes, infrequently (static zone): large e.g. 5d, frequently (dynamic zone): small e.g. 30m
- RRs that leak information should be avoided if possible
- limit information exposure by partitioning of zone files (*split* DNS)
  - two *views*: inside and outside (firewall)
  - drawback: remote users (road warriors) may not be able to use internal servers
- limit information exposure by using separate DNS servers

DNS query/response: Threats

- Forged or Bogus response
  - a compromised authoritative name server
    - compromised host
    - vulnerable DNS software
  - a poisoned cache of a resolving name server
    - spoofed response from attacker with authoritative name server IP address
    - responses from a compromised authoritative name server
  - impact: DoS, client redirection
- Removal of some RRs
  - the adversary removes part of response
  - impact: DoS

DNS query/response: Protective Measures

- underlying problem with threats to DNS query/response is integrity
- to prevent such attacks each response must be verified
  - integrity of the response
  - authenticity of the source
- achieved through DNSSEC
  - authenticate the source: start from a trusted zone such as a root zone and verify the chain down to current zone
    - public key of the trusted zone is *trust anchor*
    - the goal is to verify the public key of the source (of the response)
  - verify the response
    - the response in addition to RRs has an *authenticator*
    - the authenticator is the digital signature of the RRSet of type `RRSIG`
    - the client verifies the signature using public key of the (response) source
- to protect against removal of RR
  - `NSEC` or `NSEC3` RR is used which lists the RRTypes associated with an owner name and next name in the zone
    - this special RR is also signed and is sent to the resolving name server

DNS Zone Transfer: Protective Measures

- as discussed before is susceptible to DoS
  - limit zone transfer to a set of known entities
    - use of IP address as identity is still vulnerable to IP spoofing
- also vulnerable to tampering
- protected using *transaction signature* or `TSIG`
  - mutual identification of servers based on a shared secret
    - an adequate method: since number of servers within an administrative domain is limited
  - shared secret is also used to protect/verify the zone transfer transaction
  - a DNSSEC-signed zone only protects RRsets and not all information in a zone file
- another method to protect zone transfer is referred to as `SIG(0)`
  - similar to `RRSIG`
  - uses public key cryptography (hence more expensive)
- lower layer protection such as IPsec can also be used

Dynamic Updates: Threats and Countermeasures

Threats

- unauthorised updates
  - adding illegitimate resources to a valid zone
  - deleting legitimate resources
  - altering delegation information
    - `NS` RR to a child zone
- tampering with the dynamic update content
- replay of captured dynamic updates

Countermeasures

- authentication of entities prevent unauthorised updates and tampering with updates
  - `TSIG` and `SIG(0)` mechanisms can provide protection
    - `SIG(0)` in this case is a better choice
  - including a timestamp in dynamic update limits the replay attack
- use of lower layer protection such as IPsec

References

The materials in this document are reproduced, at times without modification, from the following sources:

- RFC 4880: OpenPGP Message Format
- "S/MIME version 3.2"
- Cryptographic Message Syntax
- RFC 6376: Domain Key Identified Mail (DKIM)
- RFC 4251: SSH Protocol Architecture
- RFC 4253: SSH Transport Layer Protocol
- RFC 4252: SSH Authentication Protocol
- RFC 4254: SSH Connection Protocol
- Bellare, Mihir, Tadayoshi Kohno, and Chanathip Namprempre. "Authenticated encryption in SSH: provably fixing the SSH binary packet protocol." Proceedings of the 9th ACM conference on Computer and communications security. ACM, 2002.
- RFC 2845: Secret Key Transaction Authentication for DNS (TSIG)

Appendix: SSH Transport Layer Protocol Key Derivation

- Output of Kex is a shared secret `K` and an exchange hash `H`
  - `H` is the hash of the first key exchange
  - `H` is also used as session identifier (will not change even if keys later change)
- Initialisation vectors:

```
IVcs = Hash(K || H || "A" || session_id)
IVsc = Hash(K || H || "B" || session_id)
```

- Encryption keys

```
KEcs = Hash(K || H || "C" || session_id)
KEsc = Hash(K || H || "D" || session_id)
```

- MAC keys

```
KIcs = Hash(K || H || "E" || session_id)
KIsc = Hash(K || H || "F" || session_id)
```

- if more bits are needed than the hash output then the process is repeated