The Design of a Cryptographic Security Architecture

Peter Gutmann

University of Auckland, New Zealand

The Problem...

…design a versatile, multiplatform, crypto architecture

Standard environment considerations

- 16/32/64 bit big/little endian CPU architecture
- Single vs multithreaded environments
- Random number generation (see Usenix Security'98)
- Data remanence problems (see Usenix Security'96)

Unusual environment considerations

- No I/O (IMS)
- It's I/O Jim, but not as we know it (VM/CMS, MVS, IBM 4758)
- Very little memory (ATM modules)
- No memory management (EMS)

Existing Approaches

Most existing approaches specify an API, not an architecture design

Designs range from very basic…

- libdes, Fortezza cryptologic interface

…to very complex…

- BSAFE, Cryptoki/PKCS #11, JCE, MS CryptoAPI v1

…are often nonportable…

- MS CryptoAPI v2

…or specific to a particular type of application…

- GSSAPI, OSF DCE Security API, SESAME

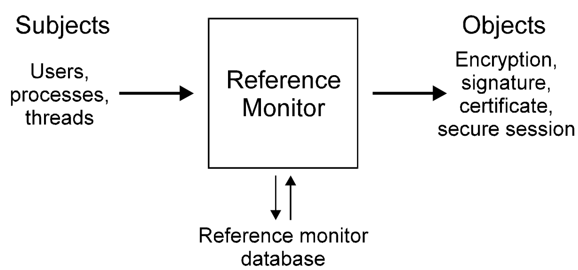
…or unmanageably large and complex

- CDSA, the emacs of crypto API's

The Solution

Architecture is built on two main concepts

- Objects encapsulate the architecture's functionality
- A security kernel enforces a consistent security policy



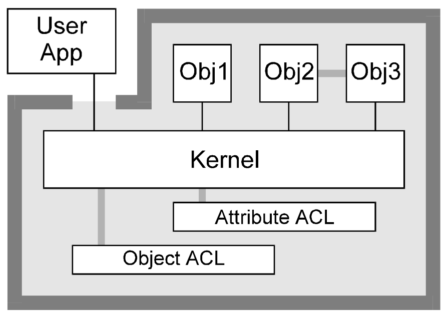
Security policy defines all permissible modes of access to objects by subjects

- Nondiscretionary policy is imposed on all subjects
- Discretionary policy can be specified by a subject and/or object

The Solution (ctd)

Properties of the architecture

- All objects are contained within the architectures security perimeter
- Kernel manages access control lists for
  - Each object
  - Each attribute read/written/deleted for each object

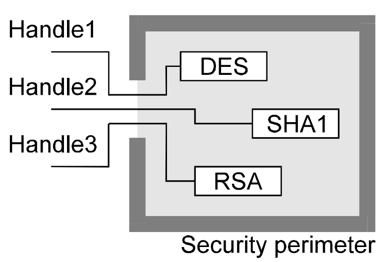


The Object Model

Two object types

- Container objects store data, keys, certificates
- Action objects perform an action on data (encrypt, sign, etc)

Objects are contained inside the architecture’s security perimeter and referenced through abstract handles

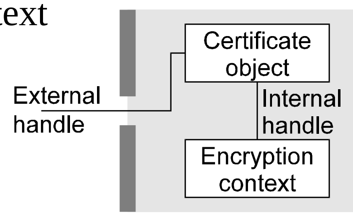


Action Objects

Encryption contexts encapsulate the functionality of a security algorithm

- DES object
- RSA object
- SHA-1 object
- HMAC-SHA object

Often associated with another object, eg public key context with certificate



Data Containers

Envelope and session objects modify the data they contain

- Type of processing is controlled by attributes set by the subject
- Resulting data format is controlled by attributes set by the subject

Usage example

```
create envelope
add signature-key attribute
push in data
pop out signed data
destroy envelope
```

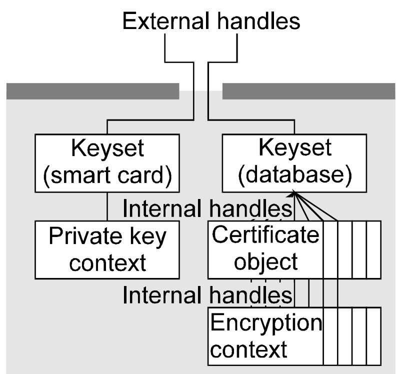
Typical envelope object use: S/MIME, PGP

Typical session object use: ssh, SSL

Key and Certificate Containers

Contain one or more keys, certificates, CRL's, etc

Appear as a (often large) collection of encryption contexts or certificate objects



Key and Certificate Containers (ctd)

Typical keysets

- Flat files with encrypted private keys
- PGP keyrings
- Smart card with public/private keys
- PKCS #11 device with keys or certificates
- Fortezza cards
- Relational database for certificates/CRL's
- LDAP directory
- HTTP for certificates/CRL's published on web pages

Security Attribute Containers

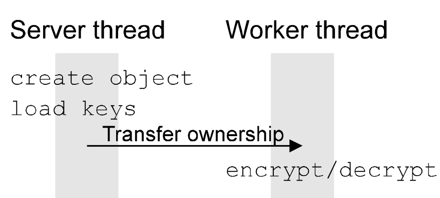
Contain attributes attached to other objects

- Certificates associated with public/private key contexts
- Certificate chains
- Signing attributes associated with envelopes

Object Security

Example use of object security

- Server thread initialises object, loads keys
- Sets forwarding count to 1, locks object

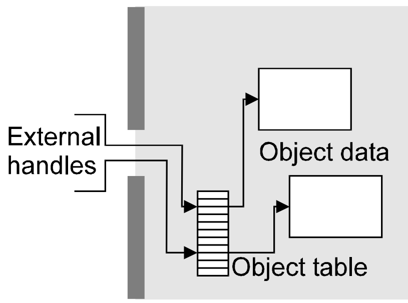


- Forwards object (changes object owner) to worker thread
  – Worker can't forward it further
  – Worker can't reload keys or change other properties
  – Original owner could also restrict usage, eg to decrypt-only

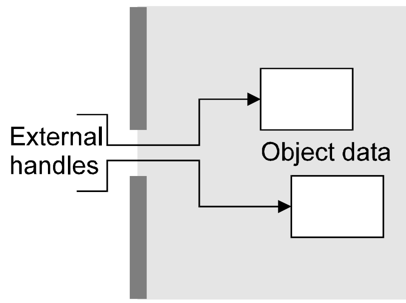
Object Access

Mandatory vs discretionary ACL checking

- ACL is enforced by kernel according to a systemwide policy
- DACL is enforced on a per-object basis



Step 1: ACL check



Step 2: DACL check

Object Access (ctd)

Objects also have object-specific discretionary ACL's

- Is the access valid for the object in its current state?

Example: Adding a subject name attribute to a certificate object is valid iff

- Size and type of attribute are valid
- Attribute is not already present
- Certificate isn't signed (and therefore immutable)

DACL checking is performed by object-specific code

Object Attribute Security

Object attributes have their own ACL's

Example attribute: Triple DES key

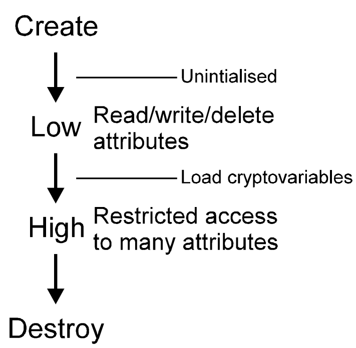
```
attribute label = CRYPT_CTXINFO_KEY
type = octet string
permissions = write-once
size = 192 bits min…192 bits max
```

Kernel checks all data passing in and out of the architecture

Attribute ACL's allow a system-wide security policy to be set

- Example: Require that CRYPT_CTXINFO_KEY can never be < 128 bits
- Even if RC2/40 or DES are present, kernel will never allow them to be used

The Object Life Cycle



Object state is changed by the kernel when a trigger event is handled

- Loading keys into an encryption context, envelope, or session object
- Signing a certificate object

Multilevel Object Security

Objects can allow different operations at different security levels

Example: Plaintext = TS, ciphertext = U

subject1	`create envelope`
	`push public key`
	`push TS plaintext`

subject2	`pop U ciphertext` `destroy envelope`
subject1'	`create deenvelope` `push TS private key`
subject2'	`push U ciphertext`
subject1'	`pop TS plaintext` `destroy envelope`

Disclaimer: Representative example only

Kernel Design

All critical security controls are enforced by the kernel

- Advantage: Security functionality is centralised
- Disadvantage: Security functionality is centralised

→ Make sure the kernel works as required

Build the kernel using good software engineering principles

- Decompose functionality into single-purpose, easy-to-understand functions
- Apply "Design by Contract"
  - Preconditions: Input conditions, assertions which are true on function entry
  - Postconditions: Output conditions, assertions which are true on function exit

Kernel Design (ctd)

C is rather limited in terms of what it can support

Use tools like ADL (Assertion Definition Language) to verify code

- Write formal spec in ADL
- Mechanical verifier checks ADL specification against implementation
- Verifier produces test documentation in quantities appropriate for ISO 9000

Kernel Design (ctd)

ADL partial example: Create a new object

```
module kernel {
  int objectTable[];
    nld { objectTable = “kernel object table” }
  int krnlCreateObject( const OBJECT_TYPE type,
                        const int objectSize )
  semantics {
    exception := cryptStatusError( return ),
    normal := !exception,
    @memfree() < objectSize <:>
      return == CRYPT_ERROR_MEMORY,
    exception --> unchanged( objectTable ),
    normally {
      isValidObject( return ),
      isInternal( return )
      }
    }
  }
```

Interobject Communications

Objects communicate via message-passing

Example: Load a key

```
msg.source:    Subject (thread/process/user)
msg.target:    Encryption context object
msg.type:      Write attribute
msg.data:      Attribute, type = Key, value = …
```

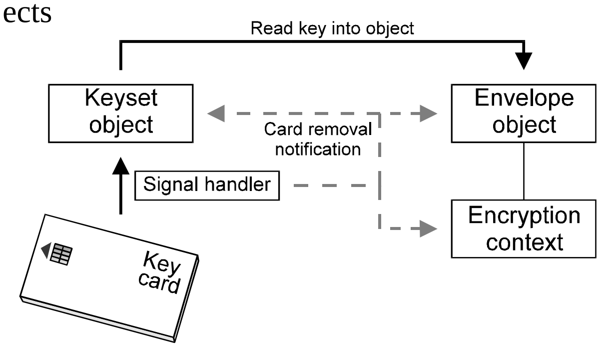
- Kernel checks the target object's ACL
- Kernel checks the attribute's ACL
- Kernel forwards message to target object

Interobject Communications (ctd)

Messages can also act as general event notifications

Example: Encryption context created from a key on a smart card

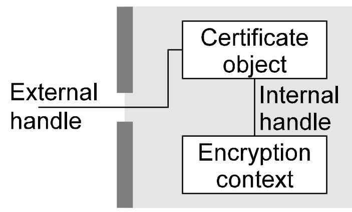
- Smart card is removed from reader, sends notification to all objects



Message Routing

Kernel routes messages to the appropriate target based on message type

Example: Message sent to certificate+context pair



- "Read validity period attribute" is forwarded to certificate
- "Read key size attribute" is forwarded to context

Message Routing (ctd)

Message routing leads to a very natural interface

- Caller need never be aware of the existence of multiple internal objects
- An object will appear to Do The Right Thing in response to a message

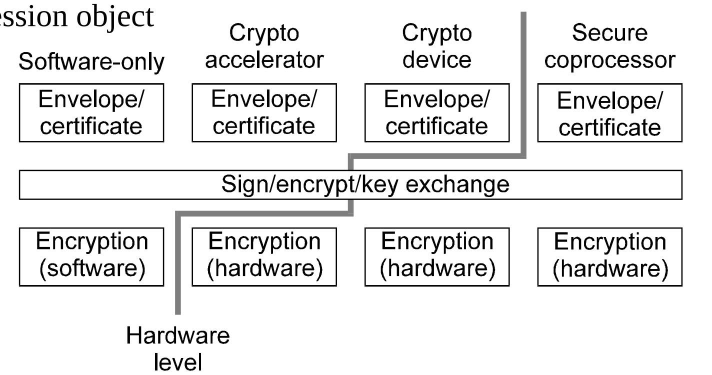
Downside: You need to re-educate users who are used to more primitive interfaces

- How do I convert a certificate into a key?
- How do I find the key size used to secure an S/MIME message (processed via an envelope)?
- How do I encrypt a message using someone's certificate?

Object Internals

Architecture design allows various levels of functionality to be encapsulated in separate modules and/or hardware

- Crypto accelerator → encryption contexts
- Crypto device (eg PKCS #11) → basic sign/encrypt level
- Secure coprocessor (eg IBM 4758) → certificate/envelope/ session object



Object Internal Details

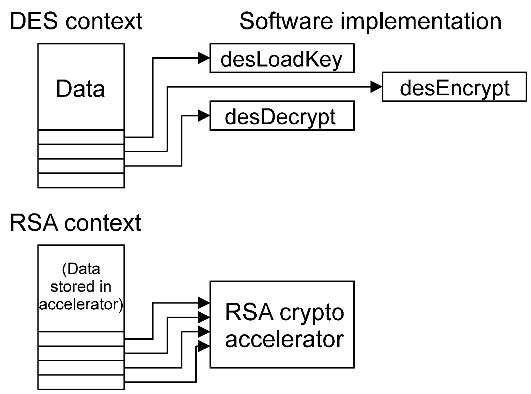
Each object consists of three main parts

- Object state information
- Message handler
- Function pointers for object methods

Example:

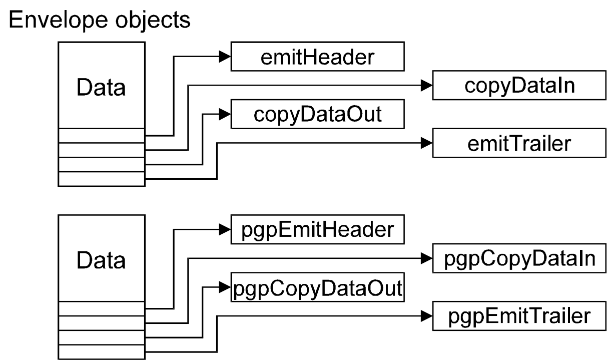
Software DES

Hardware RSA



Data Formats

Container object methods are set to format-specific functions on object creation



- To the user, the interface is identical for different output types — an enveloped message can be switched from PGP to S/MIME just by setting the envelope type on creation

Conclusion