Cincom®

CINCOM SMALLTALK™ SECURITY UPDATE

By Jerry Kott


Cincom

Motivation

- Security is **increasingly important**
- Security can be overwhelming
- Good security is invisible
- Different components used in different context
- It helps to have a big-picture view
- Know where to find stuff

Security Components

- Modular arithmetic
- Hashing & Message Digests
- Secure Randomness
- Encryption
- Key exchange

- Secure protocols - higher level, combination of the other components.
- Focus on SSL/TLS.

Modular Arithmetic

- At the core of all crypto. Finite fields arithmetic.
- Modular operations:
  - addition / subtraction
  - multiplication
  - exponentiation
  - multiplicative inverse (not simply a division)
- Operations on arbitrarily sized **LargePositiveInteger**
- **Where**: Extensions of **Integer** classes in **SecurityBase** parcel

e.g., 128-bit modular exponentiation:

16rFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
raisedTo:
16rEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
modulo:
16r0123456789ABCDEF0123456789ABCDEF

- Smalltalk native implementations
  - Classes: **MD5**, **SHA** (a.k.a. SHA-1), **SHA256**
  - Where: **SHA** & **MD5** parcels
- External libraries: **libcrypto** (OpenSsl), **BCrypt.dll** on Windows
  - Higher-bit hashing (SHA-384, SHA-512)
  - Classes: **LibCryptoHash** & **BCryptHash**
  - Where: **Xtreams-Crypto** parcel

Hashing, cont'd

- Security.Hash - outdated, we may deprecate it
- **Xtreams.Hash** - correct class to use

```
hash := Xtreams.Hash algorithm: 'sha1'.
[hash updateFrom: 'Message in the bottle!'.
digest := hash finish] ensure: [hash release].
digest.
```

```
#[87 0E 96 1F 75 02 A1 89 82 0C 7E 13 16 2C 6B 28 8A F8 A5 2A] (a ByteArray)
```

Hashing, cont'd

- Hashing streams: **Xtreams.DigestWriteStream** & **Xtreams.DigestReadStream**
- Writing accessed via the **#hashing:** method on **Xtreams.WriteStream**

```
(ByteArray new writing hashing: 'sha1')
write: 'Message ' asByteArray;
write: 'in a bottle ' asByteArray;
close;
digest
```

- Reading: API parallel to #writing on **Xtreams.ReadStream**

```
('Message in the bottle!' asByteArray reading hashing: 'sha1') rest;
close; digest
```

Secure Randomness

- Essential for generating private keys and key exchange parameters.
- Modern *nix include **/dev/urandom** - unlimited, non-blocking source of random bits. Windows API: **CryptGenRandom**
- OS-specific random source encapsulated and accessed through **DSSRandom** class, e.g.:

  DSSRandom new next
- Where: **CiphersBase** parcel

Encryption

- Symmetric - using same key for encryption & decryption:
  - **AES**, **DES**, **ARC4**, **Blowfish**
  - Where: classes & parcels with the names above
- Public/Private Key:
  - **RSA**, **DSA**, **DH**, **ECDSA, ECDH** (new in VW 8.1)
  - Where: RSA, DSA, DH, **ECC** parcels

- **ECC**: Elliptic Curve Cryptography - more security per bit of key size than RSA, more efficient
- Work in progress:
  - Native implementation 256-bit key size
  - Larger keys require SHA-384 and SHA-512 which are not yet Smalltalk native
  - External library: **libcrypto** only. BCrypt planned for VW 8.2

ECDSA

- **E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm

```
alice := PrivateKey algorithm: #ECDSA size: 256.
message := 'A signed message' asByteArray.
signature := alice sign: message.
bob := alice asPublicKey.
bob verify: signature of: message.
```

- Where: **ECC** parcel

Key Exchange

- DH (**D**iffie **H**ellman)
  - Allows computation of a shared secret from private and public keys
  - Smalltalk **native** since VW 7.1
  - **libcrypto** and **BCrypt.dll** since VW 7.9
  - Used (among others) in SSL/TLS handshake - client and server calculate a shared secret from agreed upon key generation parameters.

ECDH

- **E**lliptic **C**urve **D**iffie-**H**ellman
- Similar to DH but using Elliptic Curve Cryptography instead.
- In Cincom® VisualWorks® 8.1, it is supported in TLSv1 and higher on platforms with OpenSSL 1.0 and higher.
- ECDH Windows support coming in VW 8.2

ECDH, cont'd

```
sPrivate := PrivateKey algorithm: #ECDH size: 256.
sPublic := sPrivate asPublicKey.

cPrivate := PrivateKey algorithm: #ECDH elements: sPrivate elements.
cPublic := cPrivate asPublicKey.

sShared := sPrivate derive: cPublic.
cShared := cPrivate derive: sPublic.
```

SSL and TLS

- SSL has been supported in VisualWorks since version 5i
- **SSLv2** - a number of issues, we don't support it.
- **SSLv3** - replaced SSLv2 in 1996. Still in use but officially broken with the POODLE attack in late 2014
- In VW8.1, SSLv3 is supported but a resumable exception (**TLSInsecureProtocolWarning**) is raised during connection handshake.
- Plan to deprecate SSLv3 in VisualWorks 8.2, completely remove in a subsequent release

TLS (Transport Layer Security)

- Successor to SSL. TLSv1 would have been SSLv3.1
- TLS support since VisualWorks 7.9 (2012)
- Based on Xtreams
- **TLSv1**, **TLSv1.1**, **TLSv1.2** all supported
- Integrated into **SiouX HTTP Server**

TLS, cont‘d

- Client authentication added in VisualWorks 8.0

  (**CertificateRequest** & **CertificateVerify**)
- Support for **renegotiation_info** extension in VisualWorks 8.1.
- Growing number of cipher suites supported in each release
- e.g., **ECDHE** cipher suites added in VisualWorks 8.1

X509 Certificates

- External files (*.crt, *.key, *.pem). File extensions don't necessarily indicate certain format, potentially confusing. No real standard.
- We support PEM (base64 encoding of the certificate DER data) but it may be in a *.CRT file
- Class: **X509.Certificate**
- Where: **X509** parcel

Certificates, cont‘d

- **CertificateFileReader** - utility class to read private keys and certificates from a file, e.g.:

```
CertificateFileReader readFromFile: ‘ca-bundle.crt’
```

- Where: **X509** parcel

Certificate Management

- **X509Registry** - used to validate certificate chains.
- In TLS that means validating the peer certificates.
- **Client** needs to have trusted certificates to validate certificate chain received from **Server**.
- It's reverse with Mutual Authentication (still rare). No need to be concerned with chain validation on servers without MASSL.
- Where: **X509** parcel.

Certificate Management, cont‘d

- **TLSCertificateStore**
- Manages **owned**, **known** and **trusted** certificates
- Included in **TLSContext** for a particular client or server.
- TLSContext is a ‘root’ object for a particular client/server TLS configuration.
- Where: **TLS** parcel

Example TLSContext

- Putting it together in one of many possible ways:

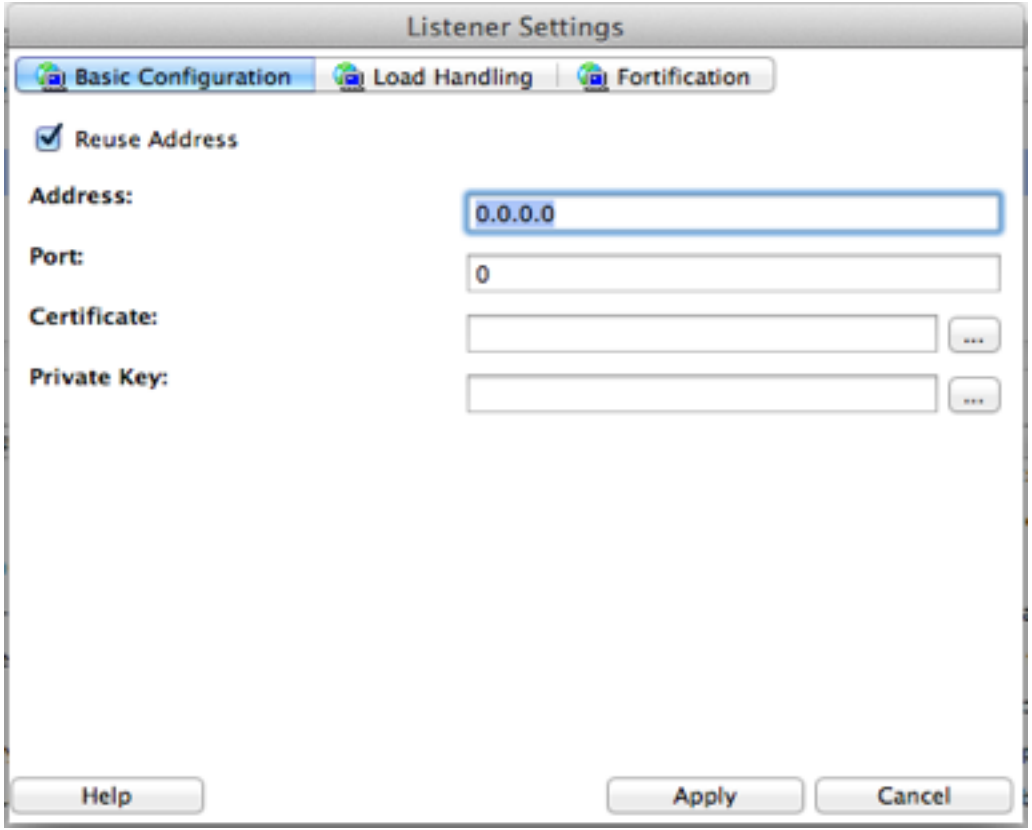
```
context := TLSContext newServerWithDefaults.
store := context certificates.
key := CertificateFileReader readFrom: ‘private.key’.
chain := CertificateFileReader readFrom: ‘chain.crt’.
store certificate: chain key: key.
```

SiouX HTTPS Configuration

```
server := Server id: ‘My Server'.
listener := server listenOn: 4433 for: HttpsConnection.
responder := Hello new.
server addResponder: responder.
server
        addSecureListener: listener
        certificateFile: ‘certificates.crt'
        privateKeyFile: ‘private.key'.
server start.
```

SiouX HTTPS Configuration

Demo:



Questions?

Contact Information

Star Team (Smalltalk Strategic Resources)

- **Suzanne Fortman** (sfortman@cincom.com)
  Cincom Smalltalk Program Director
- **Arden Thomas** (athomas@cincom.com)
  Cincom Smalltalk Product Manager
- **Jeremy Jordan** (jjordan@cincom.com)*Cincom Smalltalk Marketing Manager*
- **Suzanne Fortman** (sfortman@cincom.com)
  Cincom Smalltalk Engineering Manager

Try Cincom Smalltalk

Evaluate Cincom Smalltalk:

- **try.cincomsmalltalk.com**

Join our Cincom Smalltalk Developer Program:

- **develop.cincomsmalltalk.com**