

ENPM 809W

Introduction to Secure Software Engineering

Gananand Kini

Lecture 6

Cryptography related security bugs - Defenses



Outline



- **Securing data flows**
 - Data encryption at rest and in-transit
 - Secure password storage (Adding flavor with SALT on both client and server)
 - Secrets storage (SecureString in .NET, alternatives in Java? Python?)
 - Managing PKI and digital certificates (Certificate Transparency)
- **Weaknesses associated with cryptography implementation**
 - Cryptographic secure random number generation
 - Key derivation
 - Defending against timing attacks
- **CPSA**

Securing data flows

Securing Data Flows



- **Software systems need to protect data flows.**
- **They also need to protect data at rest.**
- **Threat modeling was done using data flow diagrams.**
- **Good time to start looking at how to protect the data sources, sinks and flows.**
- **Also look at use and misuse cases to determine:**
 - Which data needs protection? Example: passwords
 - What are the protection requirements? Example: confidential user profile data should not be exposed, user should not be able to access another user's profile information etc.

Weaknesses associated with cryptography application and implementation

Avoid ...



- **Using unknown or rolling your own cryptographic algorithm.**
- **Using weak cryptographic primitives (both for hashing or encryption/decryption).**
- **Going overboard with security because it will certainly hamper performance and usability.**

Time of Check Time of Use problems in cryptography



- **Ensure transactional checking of:**
 - Identity
 - Signatures
- **And only use the content that was checked!**

Some hash collision countermeasures



- **Randomized hash function**

- Typically best approach, but adds overhead, makes non-deterministic, & in many cases must be done at language implementation level
- “Universal hash function” is an especially efficient hash function for use in adverse environment by a language infrastructure
- Cryptographic hashes typically don’t help by themselves

If map to small number of buckets, attacker can still just precompute

- **Use another data structure (e.g., treemaps) for mapping user data in app that isn’t vulnerable to a predictable worst-case situation**
- **Limit the number of worst-case situations in each HTTP request**
 - Limit the number of different HTTP request parameters
 - Limit HTTP POST and GET request lengths
- **Limit max CPU time/request**
- **Add “random” data to value to be hashed when writing app**
 - Hand-creates randomness, but extra work for developers
 - If you’re using Java HashMap or String.hashCode, consider this

Notes on Secure Password Storage



- From a cryptographic perspective, selection of a strong hash function for storing passwords is a must.
- Use of a unique salt or a secure random value that cannot be guessed should be used in order to prevent rainbow table types of attacks.
- We will discuss more on secure password storage in the section on Authentication and Authorization.

Random Number generation pitfalls (To avoid)



- **Use of predictable random number generators for cryptography is bad**
 - C: `rand()` – **DON'T USE**
 - Java: `java.util.Random()` – **DON'T USE**
- **Forgetting to seed the random number generator -or- Using the same seed every time will generate identical sequences of numbers is bad**
- **Instead of `java.util.Random()` use `java.security.SecureRandom()`**
 - (typically) uses the SHA1PRNG generator
 - seeds itself using `/dev/urandom`
 - collects random data from disk reads, mouse movement, keystrokes, etc.
 - be careful overriding the PRNG or seed, make sure you know what you are doing

.NET Random number generation

- .NET provides [System.Security.Cryptography.RandomNumberGenerator](#) class.
- Provides secure random numbers for use in cryptographic algorithms.
- Operations
 - Fill(Span<Byte>)
 - GetBytes(Span<Byte>|Byte[])
- **Use the right function! Predictable RNG vs. Crypto RNG:**
 - Java: Random vs. SecureRandom
 - C#: System.Random vs. System.Security.Cryptography.RandomNumberGenerator
 - Python: random vs. os.urandom
 - Javascript: Math.random vs. window.crypto.getRandomValues

Use secure initialization vectors (IV)



- **Never reuse IVs**
 - Create new IV each session/encryption
 - In general, don't reuse values for crypto algorithms unless it is *specifically* documented as being fine
- **Use cryptographically random values**
 - For example, from a cryptographic pseudo-random number generator (PRNG)
 - In general, don't use non-cryptographic random values whenever using crypto algorithms

.NET Ephemeral String Storage

- **You don't want to keep cryptographic material around in memory:**
 - Use [System.Security.SecureString](#) class
 - Really there to prevent you from injuring yourself while handling passwords.
 - Does **not** secure the string in memory!
 - Provides an immutable String storage space and makes sure the object gets disposed after use.
 - Idea is to accept input directly into a SecureString and then Clear() after use.
- **Operations:**
 - AppendChar(Char) //Add to the string
 - SetAt(Int32, Char) // to modify only if necessary
 - Clear() // Clears the memory of all the values stored there
 - Dispose() // Frees up memory

Timing attacks: Solutions



- **Use timing-independent (sometimes called “constant-time”) algorithms**

- *Not the same as $O(1)$ algorithms!!*

- **E.g., “is equal to” in Java [Lawson2010] [Hale2009]:**

```
public static boolean isEqual(byte[] a, byte[] b) {  
    if (a.length != b.length) { return false; } // Omitting this is a bad bug  
    int result = 0;  
    for (int i = 0; i < a.length; i++) { // Note that we always examine everything  
        result |= a[i] ^ b[i]; // Do not use +... not constant-time on some systems  
    }  
    return result == 0;  
}
```

- **Crypto library implementations have to worry about this broadly**

- **Consider if you need timing-independent, *especially* when using crypto**

- Use them when comparing secret values (e.g., session values) where an attacker could incrementally guess a next value
- Not a problem for iterated salted hashes (too slow for attacker to get value)
- Timing attacks apply broadly, but “is equal to” is especially common issue
- MessageDigest.equal() was fixed in Java SE 6 Update 17

Credit: Prof. David A. Wheeler

Cryptographic Protocol Shapes Analyzer (CPSA)

Cryptographic Protocols Shapes Analyzer (CPSA)



- **Created at MITRE by Moses D. Liskov, John D. Ramsdell, Joshua D. Guttman, and Paul D. Rowe.**
- **Assists in the design of cryptographic protocols given:**
 - Protocol definition
 - Partial description of an execution
- **Only tries to show executions that are different.**
- **Each execution is called a shape.**
- **Helps to identify where cryptographic protocols could go wrong.**
- **Can use the Dolev-Yao model for cryptographic analysis and underlying algebra.**

Sources:

1. M. Liskov, J. Ramsdell, J. Guttman, and P. Rowe. Cryptographic Protocol Shapes Analyzer: A Manual v3. June 24, 2016. Retrieved July 20, 2021.

CPSA Quick Tutorial

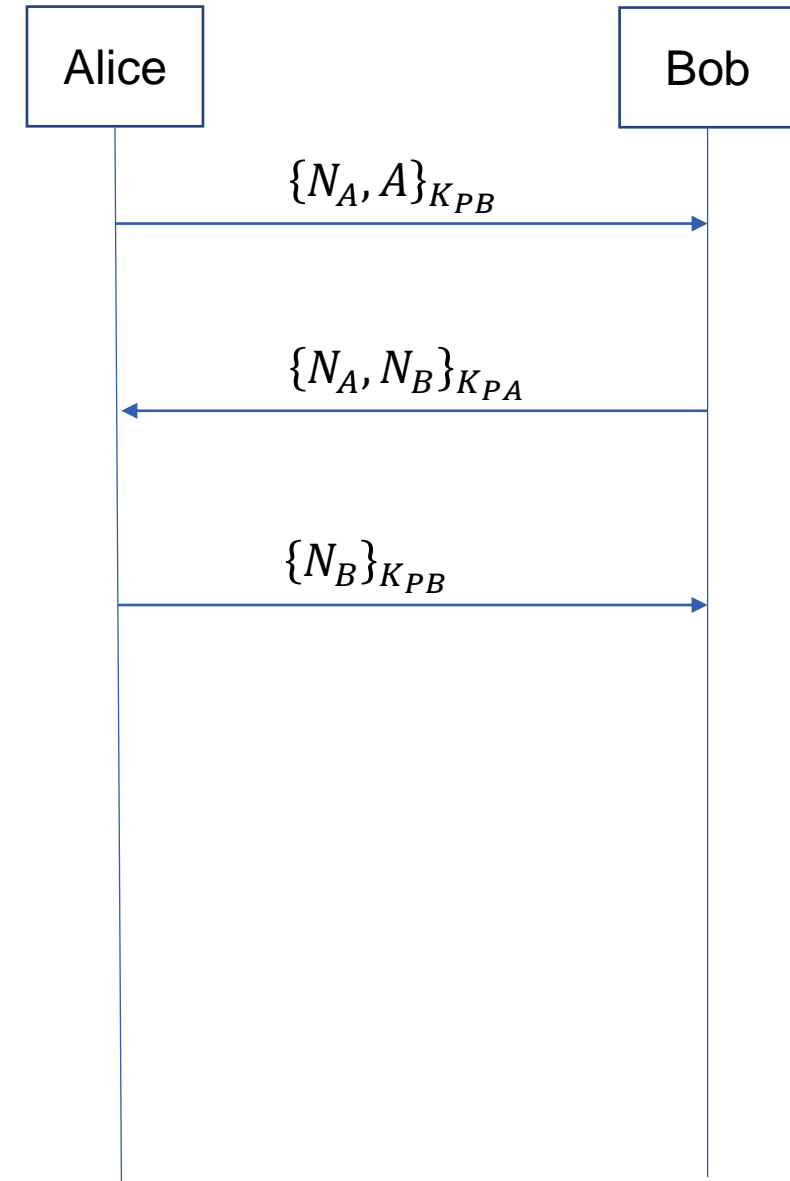


■ How do you use CPSA?

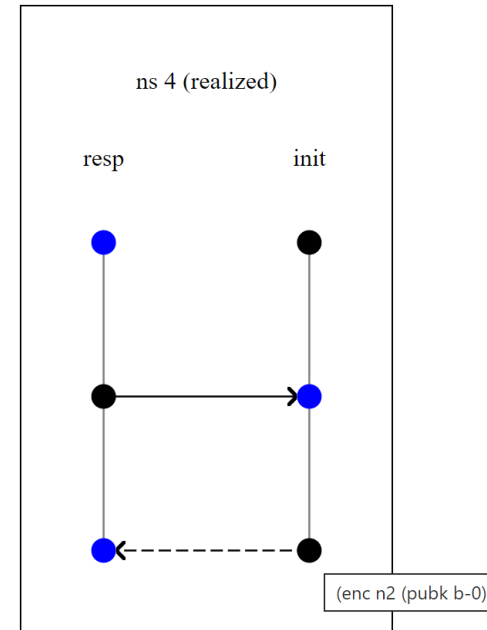
- Do Model roles that are involved in the protocol not individual actors.
 - So CPSA ignores that a message is supposed to be received by a recipient and models the network itself as an attacker. Since the attacker is free to ignore the message, the intended recipient might never get the message.
- Similarly, the intended recipient does not have to respond to messages and so you do not model specific responses.
- Do Model messages using cryptographic materials such as keys, nonces, cryptographic operations
- Do Create a skeleton that describes a partial execution of a protocol.
 - A skeleton has instances (viewpoint of honest parties assuming the role along with variable values) of the roles modeled earlier.

Needham-Schroder Protocol

- Alice and Bob want to exchange secrets over an insecure network.
- They want to establish a session key that encrypts and decrypts only their communications. How do they do it?
- K_{PA} is Alice's Public Key, K_{SA} is Alice's Secret Key
- K_{PB} is Bob's Public Key, and K_{SB} is Bob's Secret Key



Needham-Schroder MITM attack



```
(defskeleton ns
  (vars (n2 n1 text) (a b b-0 name))
  (defstrand resp 3 (n2 n2) (n1 n1) (b b) (a a))
  (defstrand init 3 (n1 n1) (n2 n2) (a a) (b b-0))
  (precedes ((0 1) (1 1)) ((1 2) (0 2)))
  (non-orig (privk a) (privk b))
  (uniq-orig n2)
  (operation nonce-test (added-strand init 3) n2 (0 2)
    (enc n1 n2 (pubk a)))
  (label 4)
  (parent 3)
  (unrealized)
  (shape)
  (maps ((0) ((a a) (b b) (n2 n2) (n1 n1))))
  (origs (n2 (0 1))))
```

Needham-Schroder in CPSA



- CPSA uses a `herald` for title of the code.
- The comment is shown below the title.
- Three semicolons also are used for inline comments.
- Define protocol using `defprotocol`
- Define roles using `defrole`
 - In this case initiator (`init`) and responder (`resp`)
- Define a skeleton containing “strands” using `defskeleton`.
 - Strands are S-expressions (Lisp) that define an instance of the protocol from the provided role’s view along with some “maplets” or variable values to use for various parts of the protocol.

```
(herald "Needham-Schroeder Public-Key Protocol"
      (comment "This protocol contains a man-in-the-middle"
               "attack discovered by Galvin Lowe."))

;;; Used to generate output for inclusion in the primer.
;;; Use margin = 60 (-m 60) to generate the output.

(defprotocol ns basic
  (defrole init
    (vars (a b name) (n1 n2 text))
    (trace
     (send (enc n1 a (pubk b)))
     (recv (enc n1 n2 (pubk a)))
     (send (enc n2 (pubk b)))))
  (defrole resp
    (vars (b a name) (n2 n1 text))
    (trace
     (recv (enc n1 a (pubk b)))
     (send (enc n1 n2 (pubk a)))
     (recv (enc n2 (pubk b)))))
    (comment "Needham-Schroeder"))

;;; The initiator point-of-view
(defskeleton ns
  (vars (a b name) (n1 text))
  (defstrand init 3 (a a) (b b) (n1 n1))
  (non-orig (privk b) (privk a))
  (uniq-orig n1)
  (comment "Initiator point-of-view"))

;;; The responder point-of-view
(defskeleton ns
  (vars (a b name) (n2 text))
  (defstrand resp 3 (a a) (b b) (n2 n2))
  (non-orig (privk a) (privk b))
  (uniq-orig n2)
  (comment "Responder point-of-view"))
```

Maplet (role name, role value)

Next time ...



- **Authentication and Authorization related bugs - Attacks**