CSC 580
Cryptography and Computer Security

Authenticated Encryption, Key Wrapping, and PRNGs
(Sections 12.6-12.9)

April 3, 2018
Goal: Protect both Confidentiality and Integrity

Some techniques that have been used:

- Encrypt with hash of message: $E(K, M \ || \ H(M) )$
  - $E$ better be non-malleable!! (problem with WEP using RC4)

- Encrypt with MAC: $E(K_1, M \ || \ MAC(K_2, M))$
  - Used in SSL/TLS

- Encrypt followed by MAC: $C = E(K_1, M) \ ; \ T = MAC(K_2, C)$
  - Used in IPSec

- Encrypt and MAC: $C = E(K_1, M) \ ; \ T = MAC(K_2, M)$
  - Used in SSH

Notes:

- Important to use different keys for encryption and MAC (avoid interactions)
- All techniques have drawbacks
New and Improved! Authenticated Encryption
High-Level Idea

Ideas:
- Design for confidentiality and integrity together - use a single key!
- Allow some data to be transmitted in the clear, but still authenticated

![Diagram of Authenticated Encryption (AE)]
**JCA - Using Authenticated Encryption**

**Example using GCM (one AE mode)**

```java
GCMParameterSpec s = ...;
cipher.init(..., s);

// If the GCM parameters were generated by the provider, it can
// be retrieved by:
// cipher.getParameters().getParameterSpec(GCMParameterSpec.class);

cipher.updateAAD(...); // AAD (optional - must be before plaintext)
cipher.update(...);    // Multi-part update
cipher.doFinal(...);   // conclusion of operation

// Use a different IV value for every encryption
byte[] newIv = ...;
s = new GCMParameterSpec(s.getTLen(), newIv);
cipher.init(..., s);
...
```

On encryption: Tag is embedded in output ciphertext (you don’t have to handle!)
On decryption: Bad tag results in throwing AEADBadTagException
Two AE modes: CCM and GCM

CCM (Counter with CBC-MAC)
- Ciphertext produced using CTR mode
- MAC produced using CBC-based MAC
- The good: Strong, provable security under certain assumptions
- The bad:
  - Encrypt/MAC require two independent block cipher calls
  - Inclusion of CBC means not parallelizable

GCM (Galois/Counter Mode)
- CTR mode encryption - *almost*... incr 32-bits → $2^{39}$-bit limit on size
- GHASH to auth *ciphertext* - one Galois Field (GF) mult per block
- The good:
  - Strong, provable security under certain assumptions
  - Per block: 1 block cipher call, and one GF mult (Intel instruction) - fast!
  - Block cipher calls are parallelizable (just like CTR mode)
- The bad: ?
GCM - Algorithm Overview
Hash and Encryption Functions

A little misleading: When combined, these $X_i$’s are ciphertext blocks (called $Y_i$ below)!

(a) $\text{GHASH}_H(X_1 \parallel X_2 \parallel \ldots \parallel X_m) = Y_m$

(b) $\text{GCTR}_K(\text{ICB}, X_1 \parallel X_2 \parallel \ldots \parallel X_n^o) = Y_1 \parallel Y_2 \parallel \ldots \parallel Y_n^o$

Figure 12.10 GCM Authentication and Encryption Functions
GCM - Algorithm Overview
Overall GCM operation

Figure 12.11 Galois Counter - Message Authentication Code (GCM)
Key Wrapping

Consider: In the JCA KeyStore, keys are stored in a file. How are they protected?

- Password used to “unlock” the KeyStore
- Need to use encryption with one key to encrypt another key
- An AES 256-bit key spans multiple blocks of AES
- Can a specially designed mode help?
  - Advantage: Limited size plaintext (can have all in memory at once)
  - Speed isn’t as big an issue as it is with bulk encryption
  - Wrapped key is random - how do you know decryption is right - authentication!
  - Specially designed mode: Key Wrap (KW) mode

Related notions with different terminology:

- **Key Wrapping**: Encrypting a symmetric key using symmetric cipher
- **Key Encapsulation**: Encrypting a symmetric key using a public key algorithm (e.g., for hybrid encryption)
AES Key Wrap Mode
Pseudocode from NIST publication

Inputs: Plaintext, $n$ 64-bit values $\{P_1, P_2, \ldots, P_n\}$,
Key, $K$ (the KEK).

Outputs: Ciphertext, $(n+1)$ 64-bit values $\{C_0, C_1, \ldots, C_n\}$.

1) Initialize variables
   Set $A^0 = IV$, an initial value (see 2.2.3)
   For $i = 1, \ldots, n$
   \[ R^0_i = P_i \]

2) Calculate intermediate values
   For $t = 1, \ldots, s$, where $s = 6n$
   \[ A^t = \text{MSB}_{64} \left( \text{AES}_K \left( A^{t-1} | R^{t-1}_1 \right) \right) + t \]
   For $i = 1, \ldots, n - 1$
   \[ R^t_i = R^{t-1}_{i+1} \]
   \[ R^n_t = \text{LSB}_{64} \left( \text{AES}_K \left( A^{t-1} | R^{t-1}_1 \right) \right) \]

3) Output the results
   Set $C_0 = A^t$
   For $i = 1, \ldots, n$
   \[ C_i = R^t_i \]

Default IV is hex:

Each 64-bit plaintext block gets “shifted through” encryption position 6 times.
AES Key Wrap Mode
Diagram of one stage (from NIST)
Observations:

- PRNGs need uniformly distributed output
  - Good hash functions and MACs have uniformly distributed outputs

- PRNGs need to be one-way so seed/state can’t be derived
  - Good hash functions and MACs are preimage resistant (one-way)

- PRNGs need output to be computationally uncorrelated (independent)
  - Good hash functions and MACs have collision resistance

And in addition: Hash functions and MACs tend to be fast

So…. Can we use hash functions and MACs to make good PRNGs?
PRNGs from hash functions

Idea: Concatenate seed and counter, and run through hash fn

So: Initialize $V = \text{seed} \ || \ 0$

![Diagram of PRNG using cryptographic hash function](From Figure 12.14 in the textbook)

This is essentially how the standard Java SHA1PRNG instance of SecureRandom works (generally the default)
PRNGs from MACs

Can use a simple feedback loop with a MAC (NIST SP 800-90)

(b) PRNG using HMAC

Some other options
- Can use a MAC with a counter, like previous slide (IEEE 802.11i does this)
- Can do feedback, but concatenate a constant (the seed) each iteration (TLS)