**601.445/600.454: Practical Cryptographic Systems** October 2, 2017

# Assignment 2 (Part 2)

*Instructor: Matthew Green* *Due: 11:59pm, October 10*

Name: _____

**The assignment should be completed individually. You are permitted to use the Internet and any printed references.**

**Please submit the completed assignment via Blackboard.**

**Problem 1**: Active Attacks (60 points)

In the first part of this assignment you were asked to implement a cryptographic specification. This resulted in a utility for encrypting and decrypting using a symmetric key $K$. In this part of the assignment you will develop a tool that programmatically decrypts any ciphertext produced by your encryption utility from Part 1. Your tool will not have access to the decryption key. It will instead call a second program that attempts to decrypt the ciphertext using the decryption key, and returns an error on failure.

The command line profile for your tool will be as follows:

```
decrypt-attack -i <ciphertext file>
```

This program will take as input a ciphertext encrypted with the key $K$. When it completes it should output the decryption of the ciphertext. This program will call a second program called `decrypt-test` that *has the key $K$ hardcoded into it.* The profile for `decrypt-test` will be as follows:

```
decrypt-test -i <ciphertext file>
```

The utility will have a hard-coded decryption key. It *will not return the decrypted ciphertext*, but instead only a single one of the following three response messages:

1. "SUCCESS"
2. "INVALID PADDING"
3. "INVALID MAC"

You are expected to implement the tool `decrypt-test` using your own code from Part 1 of this assignment, though you do not have to turn it in. You only need to turn in `decrypt-attack`, as we will provide our own implementation of `decrypt-test` for grading.

For test purposes you should also generate your own key $K$ (to hard-code into `decrypt-test`) and generate a test ciphertext based on some plaintext of input size at least 256 bytes.

**Problem 2**: Padding oracles in practice (10 points).

TLS and Datagram TLS (DTLS) each use a form of encryption that's nearly identical to the scheme you implemented in Problem 1. The following partial code listing is responsible for processing a received DTLS packet (source: `d1_pkt.c` in OpenSSL version 1.0.0e).

Note that this listing calls two subroutines that are *not* shown here:

1. `s->method->ssl3_enc->enc(...)` performs decryption and check the padding. This returns -1 if the padding is invalid.
2. `s->method->ssl3_enc->mac(...)` computes the MAC.

If you want to look at those subroutines you can find them in the OpenSSL codebase at openssl.org. But you shouldn't need them to answer the questions below.

```
static int
dtls1_process_record(SSL *s)
{
...
/* decrypt in place in 'rr->input' */
rr->data=rr->input;

enc_err = s->method->ssl3_enc->enc(s,0);  /* <<<--- decryption/padding check */
if (enc_err <= 0)
{
/* decryption failed, discard message */
if (enc_err < 0)
{
rr->length = 0;
s->packet_length = 0;
}
goto err;
}

...

/* r->length is now the compressed data plus mac */
if ( (sess == NULL) ||
(s->enc_read_ctx == NULL) ||
(s->read_hash == NULL))
clear=1;

if (!clear)
{
/* !clear => s->read_hash != NULL => mac_size != -1 */
int t;
t=EVP_MD_CTX_size(s->read_hash);
```

```
OPENSSL_assert(t >= 0);
mac_size=t;

if (rr->length > SSL3_RT_MAX_COMPRESSED_LENGTH+mac_size)
{
...
}
/* check the MAC for rr->input (it's in mac_size bytes at the tail) */
if (rr->length < mac_size)
{
...
}
rr->length-=mac_size;
i=s->method->ssl3_enc->mac(s,md,0);  /* <<<--- MAC computation */
if (i < 0 || memcmp(md,&(rr->data[rr->length]),mac_size) != 0)
{
goto err;
}
}

...

f_err:
ssl3_send_alert(s,SSL3_AL_FATAL,al);
err:
return(0);
}
```

1. Assume that the attacker is sending packets to the server for decryption, and that he receives some notification from the server when the routine outputs an error (*i.e.,* returns 0). How might he execute a padding oracle attack? (Hint: this is trickier than it looks!)