Cryptography CW2

January 6 2020

1 Cryptographic Software

1.1

name: duncan rowe
Command: `openssl rsautl -pubin -inkey 1apubkey.pem -encrypt -in name | base64`
Result:

```
L3qgbf/Lxl7C5Sdi9ZMH03NAjttA7+Fk1bu9bX+czaSBqa3nn4nHYo7VPj/lv9ZYhffS3h3kTVT+
bCc8xHvz2f2SYC8ny4S4UCqO1fZF9gRCbOOHWdeUXlbM9IzyxsulQDJC98HMItVA6XjMnCHchU06
il3NIvfVPavFIZJEv9x06x7LjBUEHlKHfGE4mJYJ7OptSm0KLaEjB57Gj4Fu44odR4YeTIE5GtOf
Pev7dXID0ElbZj0vBgZHeJpowYSeMRYdas1/f8Wi14a72gTNcMtQ0dGq0CYptx3PY7zAgm+/lyNf
SfTBeE4cjFSJmRqBPf+lsxj5WVnvJP7Ue71TFA==
```

1.2

generate public/private key: `openssl genpkey -paramfile 1bdhpparameters.pem -out 1bdhkey.pem`
get public key command: `openssl pkey -in 1bdhkey.pem -pubout | base64`
public key:

```
LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlJQkh6Q0JsUVlKS29aSWh2Y05BUU1CTUlHSEFv
R0JBTDBTM0JUTmhqbGN5MkNoTFdDeHlNemRNRW9DWTJNTwp1OVZteGxHb2lQY0xOa1VBbUNwdEpE
NEYxZDk1Rk9zd2dnU2dkRENUZ2FEV2l6WmFlZnloSDhYMWJZTnFOcnRVCnlaVmp3MUtqU3pTdC9D
cTBDeTdoMEJ2SXF0bWZ2aUNjQW5vdmIzQ2dBcDlXdUxtYWdydXF2K3MyL1NncG1lYkgKUUFJVy9x
bkgyRE1MQWdFQ0E0R0VBQUtCZ0RyRFdHSmRnVjRCUDVUTTFZb0xyY2JSNGpNdXhkMXlXQ0ZIY0Q4
bQpzQ2hMRjAwWU1kYTRRTUd1dDRpbUFYNnhUenNranJsVElLT00yQnppT2FEMHlWRFJyaGNKc0Vk
WGlqakM2SXFsCmJEQVF0dGt3dUMxdlhnUkNEa2hRTFA1Ky90L3J4dE5CNzF1dlprWFRDeDhRWXFj
Q0hNNFZqTkhoaWZOSStldEUKSUhBMgotLS0tLUVORCBQVUJMSUMgS0VZLS0tLS0K
```

generate shared key: `openssl pkeyutl -derive -inkey 1bdhkey.pem -peerkey 1bpubkey.pem | base64`
shared key:

```
GNsW6M2cJxQ64PuBexzJHzKItiDey3H2QOKy1WDgLu6SqM5Av4itH+1w9ZFqYM+7f576N7dOBOwN
QBBOJ46FtynINLQdxPgqczqyul19QsPYfHI96TkxJKLwQiUk/H91r+ef/5egzpNUKrJ7IiFSNLoH
XxXGWQC1odZCO4ExtOs=
```

1.3

generate RSA keys: `openssl genrsa -out 1crsa.pem 2048`
seperate public key from private: `openssl rsa -in 1crsa.pem -pubout > 1cpublic.pem`

```
-----BEGIN PUBLIC KEY-----
MIIBIDCBlQYJKoZIhvcNAQMBMIGHAoGBAL0S3BTNhjlcy2ChLWCxyMzdMEoCY2MO
u9VmxlGoiPcLNkUAmCptJD4F1d95FOswggSgdDCTgaDWizZaefyhH8X1bYNqNrtU
yZVjw1KjSzSt/Cq0Cy7h0BvIqtmfviCcAnovb3CgAp9WuLmagruqv+s2/SgpmebH
QAIW/qnH2DMLAgECA4GFAAKBgQCEUQc+0TuYPrnNP7KP/1tR1X0tV4B/GtbKrCFU
863vMPHzozchNvxz+uI2OJGet0evMnz7xCOoZ+w5fE00sldFegAwIaC4QW6BBZcO
V2Twtbggz+95r7yKfNRM2ty0SV5CE8fdCeRBoMPver9gJk4uN7KyHFXWbOUzt9YR
tiJVTQ==
-----END PUBLIC KEY-----
```

message: duncan rowe
signature command: `openssl dgst -sha256 -sign 1crsa.pem name.txt | base64`
signature:

```
SFqXVL4q05Z3+jr+7s4Ozhvk4P+9Wt+zuwsxKWIKenZG1t6kGr0yU0ICiGjzWqfdZx08FDyHDTX0
odXYZqxsx5wlQ5YcmIqwEjLsQ1uzAxcV3S37ctuoDStju22pc6LFnJKOLXIHKxYzo2/y8d4O/Q92
mvZFKgioWTc5zv4r3pq5yjSETzEBq3IRC20s++SV2MDo5y4A8u47jfXAmgboASmusA/PCG74s6SQ
82Fy2nmfobaNPjYFfysypAVOpkmgECew4Rmn7SSXGr0eOLHxPHHHxGPwDtHPE6CUKRS9EY9uWusy
YclcEr2yvr9XqEhZAjmcqKg4IjkcM8DSTpOF6w==
```

1.4

command: `openssl x509 -text -noout -in tls.crt`

1.4.1

host: cloudflare.com port: 443

1.4.2

serial number: 0f:d5:85:d4:4b:7d:b3:7f:5c:3e:65:3b:f4:b2:26:f7

1.4.3

valid end date: Jul 4 12:00:00 2021

1.4.4

algorithm used to sign: ecdsa-with-SHA256

1.4.5

fingerprint:

```
04:1b:cc:89:ac:a5:98:94:f4:06:e8:9d:ca:80:a8:
ac:e1:77:8e:63:7f:dc:2f:9a:84:43:e9:8f:df:38:
25:e3:29:62:a6:05:fb:fd:72:4b:89:0e:77:eb:56:
fc:7c:98:63:fd:48:c8:b9:d1:49:d3:12:fd:89:b3:
ab:b3:a2:53:0e
```

2 Message Authentication Codes

2.1

2.1.1

No it is not secure. First we query the oracle with a message and receive the tag $IV|T_s$. Now that we know the IV, we can do a modification of a length-extension attack where we add message blocks before our valid message such that the eventual xor before we reach the IV is equal to our original first message block. In this way, we can forge any message blocks added before our valid block and create a valid tag.

2.1.2

No it is not secure. The output is the all the previous block tags combined. However as the IV is 0, for messages of size b bits we are able to get the results of F_k. Therefore we can craft an attack where we query with M_1 which is size b bits to get the output T_1. We XOR T_1 with our message M_2 and query again, receiving tag T_2. We then combine the message $M_1 + M_2$ with the tags we received (T_1, T_2) and we have created a valid tag.

2.2

To create a hash-then-mac scheme, we are given a hash function H that takes any size input and converts it into a hash of size l. This hash is said to be collision resistant, indicating that the probability of two inputs mapping to the same hash is negligible

$$Pr[HashColl_{A,H}(\lambda) = 1] \leq negl(\lambda)$$

This hash value is then given to the MAC function provided which is stated to be EUF-CMA secure. This allows the user to authenticate messages of arbitrary length.
It is given that the MAC configuration given in the problem is EUF-CMA secure. This means that

$$Pr[Game_{(A,MAC)}(\lambda) = 1] \leq negl(\lambda)$$

In order to prove that our hash-then-mac scheme is also EUF-CMA secure we need to show that our hash probability does not add non-negligible probability to EUF-CMA.

$$Pr[HashColl_{A,H}(\lambda) = 1] + Pr[Game_{(A,MAC)}(\lambda) = 1] \leq negl(\lambda)$$

The concern is that his probability of a hash collision allows adversary A to gain an advantage in the EUF-CMA game, giving him a higher probability to forge a tag for message M as the message space for messages has been decreased. However as it has been shown that a hash collision is negligible in this message space. Furthermore the probability for a hash collision is so small that combining it with the probability for the EUF-CMA game should still result in a scheme that is EUF-CMA secure.

2.3

MAC' = (Gen',Tag',Verify') where Tag' is created by $Tag_k(m_1 m_2 ... m_i)$ where the message is b bits long.

Assuming b is at least 32 bits. This allows MAC' to be WEUF-CMA secure as the probability of

$$Pr[Game_{(A,MAC)}(\lambda) = 1] \leq negl(\lambda)$$

is negligible due to A having a 2^b chance of generating the correct tag, message combination without the verification oracle. However, if the adversary is given the verification oracle in order to be EUF-CMA secure, then this scheme is not EUF-CMA secure as it would trivial for an attacker to brute force the tag for any message.

3 Number Theory, Group Theory and Computationally Hard Problems

3.1

3.1.1

$$11^{1049} \bmod 21$$
$$21 = 3 * 7$$
$$1 = 7 - 3 * 2$$

$$11^{1049} \bmod 3$$
$$(11 \bmod 3)^{1049} \bmod 3 = 2^{1049 \bmod 2} \bmod 3 = 2$$

$$11^{1049} \bmod 7$$
$$(11 \bmod 7)^{1049} \bmod 7 = 4^{1049 \bmod 6} \bmod 7 = 4^5 \bmod 7 = 2$$

$$11^{1049} \bmod 21 = (7 * 2 - 2 * 3 * 2) \bmod 21 = 2$$

3.1.2

$$20^{179} \bmod 21$$
$$21 = 3 * 7$$
$$1 = 7 - 3 * 2$$

$$20^{179} \bmod 3$$
$$(20 \bmod 3)^{179} \bmod 3 = 2^{179 \bmod 2} \bmod 3 = 2$$

$$20^{179} \bmod 7$$
$$(20 \bmod 7)^{179} \bmod 7 = 6^{179 \bmod 6} \bmod 7 = 6^5 \bmod 7 = 6$$

$$20^{179} \bmod 21 = (7 * 2 - 6 * 3 * 2) \bmod 21 = 20$$

3.1.3

$$12^{2091} \bmod 33$$
$$33 = 3 * 11$$
$$1 = 3 * 4 - 11$$

$$12^{2091} \bmod 3$$
$$(12 \bmod 3)^{2091} \bmod 3 = 0$$

$$12^{2091} \bmod 11$$
$$(12 \bmod 11)^{2091} \bmod 11 = 1$$

$$12^{2091} \bmod 33 = (3 * 4 * 1 - 11 * 0) \bmod 33 = 12$$

3.2

3.2.1

$$\phi(97) = 96$$
$$\phi(96) = 33 * 2 = 32$$

3.2.2

The multiplicative inverse of 23 in $\mathbb{Z}_{97}^*$ is 38 as

$$97 = 23 * 4 + 5$$
$$23 = 5 * 4 + 3$$
$$5 = 3 * 1 + 2$$
$$3 = 2 * 1 + 1$$
$$1 = 23 * 38 - 9 * 97$$

3.2.3

The order of 2 for the group $\mathbb{Z}_{97}^*$ must belong in the group $(1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96)$. The only element that satisfies $2 * i \bmod 97 = e$ where e is the identity element (1 for this group) is 48.

3.2.4

Subgroups of the group $\mathbb{Z}_{97}^*$ must have an order that divides into 96. These possible orders are $1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96$. The non-trivial subgroups have orders $2, 3, 4, 6, 8, 12, 16, 24, 32, 48$. Knowing this, as we know the order also has to be a generator, we can simply try out each order as a generator and count the numbers generated. The subgroup that satisfies the two requirements is of order 24 as it contains a generator $< 24 >$ that produces a subgroup of order 24.

3.3

$$n = 38749709$$
$$(e, d) = (10988423, 16784693)$$
$$ed = 1 \bmod n = 1 + k\phi(n)$$
$$ed - 1 = k\phi(n)$$
$$184437306609139 - 1 = k\phi(n)$$
$$184437306609138 = 2^s r$$
$$r = 92218653304569$$

choosing a random x

$$x^r \bmod n = 1$$

when x is 13

$$13^{2r} = 1 \bmod n$$
$$m = 13^{3r} \bmod n = 31285298$$
$$31285298 + 1 = 1 \bmod p$$
$$31285298 - 1 = 1 \bmod q$$

$$31285299 = 1 \bmod p$$
$$31285297 = 1 \bmod q$$

Solving for p and q gives us the primes $5347, 7247$

3.4

Given n and $\phi(n)$, which represents the number of factors of integers co-prime to n. As n can not be prime $\phi(n)$ reveals information about the factors of n as $\phi(n) = \phi(p)\phi(q) = (p - 1)(q - 1)$
Given

$$\phi(n) = (p - 1)(q - 1) = pq + (-p - q) + 1$$

As pq = n

$$\phi(n) = n - p - q + 1$$
$$p + q = n + 1 - \phi(n)$$

Now consider the polynomial with roots of p and q

$$(x - p)(x - q) = x^2 - (p + q)x + pq$$

We can substitute our values above to create a polynomial whose roots are solvable in polynomial time

$$(x - p)(x - q) = x^2 - (n + 1 - \phi(n))x + n$$

4 Public-key Encryption

4.1

The adversary knows that $\pi = u^a v^b$. The value of $u^a v^b$ is uniformly distributed and the ciphertext is rejected unless $\pi = u^a v^b$. If it is rejected only one value is removed, leaving $p-1$ possibilities remaining. Continuing this way, the nth decryption will be rejected except with probability $1/(p-n+1)$.Therefore the probability it won't be rejected is $n/(p-n+1)$. Therefore as q is exponential while n is polynomial this proves that the attacker can not guess the pair with more than probability $1/p$

4.2

This security notion does not allow the attacker access to the oracle after receiving the challenge ciphertext. Furthermore, when querying the oracle, the adversary is only given the public key and not the security parameter used to generate the key. This is effectively a weaker version of IND-CCA as IND-CCA still gives the adversary access to the decryption oracle after giving the ciphertext. This makes the IND-LBA attack also IND-CPA secure as well as IND-CCA is also IND-CPA secure.

4.3

We are trying to determine the probability of a ciphertext passing the $\pi = u^a v^b$ check but not containing a valid message where $m = wu^{-x}$. As the adversary is unbounded, we assume it can compute π which constrains (a,b)

$$log_{g_1} c = a + (log_{g_1} g_2)b = a + \alpha b$$

$$log_{g_1} u = r_1$$
$$log_{g_2} v = r_2$$

$$log_{g_1} \pi = a log_{g_1} u + b log_{g_2} v$$
$$log_{g_1} \pi = r1a + \alpha r2b$$

Therefore the value of π is uniformly distributed with the adversaries probability of finding the value $1/q$. If the oracle returns with an error, then the adversary knows that the value is not q, eliminating only one of those possibilities. This probability will also hold for subsequent decryption queries as if the adversary performs p queries (assuming they are all rejected with an error) there are still q-p queries left, making the probability of a correct guess $1/(q-p)$. If a total of p queries are sent then an invalid ciphertext not being rejected has probability p/q-p. This makes the probability negligible as q is exponential and p is polynomially bound.

4.4

5 Digital Signatures

5.1

5.1.1

If n=77 then p=7 and q=11. $\phi(n) = 60$ and as ed = 1 mod 60 d can equal 11, 71 etc.

If $\delta = 3$ and $m = \delta^e$ mod N then 3^{11} mod $77 = 47$

If $m = 2$ and $\delta = m^d$ mod N then 2^{11} mod $77 = 46$

If $e = 59$ and $m = 3$ then $ed = 1$ mod 60 and $d = 59$ so $\delta = m^d$ mod N then 3^{59} mod $77 = 26$

5.1.2

square-multiply $= 1.5 * 2k * (2k)^2 = 12k^3$

$$\sigma = m^d \bmod p$$
$$\sigma = m^d \bmod q$$

$$\sigma = m^d \bmod p = (m \bmod p)^d \bmod p$$

As d is moded to k bits the length of l_d is now k. Using the equation for efficiency with this new value we get $1.5 * k * k^2$ and we do this for both p and q so the total is $3k^3$.

5.1.3

As $m = \delta^e \bmod N$ and we know the value of e, this means $m_1 = \delta^5 \bmod N_1$ which can we written as

$$\delta^5 = m_1 \bmod N_1 = m_2 \bmod N_2 = ...$$

Using the Chinese Remainder Theorem we know that as $N_1, N_2...$ are co-prime, we can multiply them all to form N. We then need to find x_i such that $n_i x_i = 1 \bmod N_i$ where $n_i = N/N_i$. Once we solve for x_i, we can sum all our values to solve for δ like this $\delta^5 = \sum m_i n_i x_i \bmod N$. In this way we can solve for δ.

5.1.4

We know that $m_1 = \sigma^{e1} \bmod n$ and that $m_2 = \sigma^{e2} \bmod n$. We also know that because the $gcd(e1, e2) = 1$ that $1 = e1x + e2y$. Knowing this we can raise the entire equation to the power of σ to get the result of $\sigma = \sigma^{e1x}\sigma^{e2y}$. We know from previous that $\sigma^{e1} = m1$ and $\sigma^{e2} = m2$ so we can substitute our values. Furthermore once we perform the euclidean theorem of e1 and e2, we will know the values of x and y and therefore be able to solve for σ.

5.2

RSA-FDH defines the signature as $\delta = H(M)^d \bmod n$. If hash collisions are found in hash function H, this means that the

$$Pr[HashColl_{A,H}(\lambda) = 1] \geq negl(\lambda)$$

This affects the EUF-CMA security for RSA-FDH. EUF-CMA security is defined as

$$Pr[Game_{(A,DS)}(\lambda) = 1] \leq negl(\lambda)$$

If an adversary is able to generate a hash with non-negligible probability they will be able to falsely generate a signature for message M_1 with another message M_2, creating the same hash where $H(M_1) = H(M_2)$. This will create the same signature value, allowing their message to be verified even though the messages are different. This breaks EUF-CMA security.