Cryptography: Public Key Cryptography

Alex Corner

Sheffield Hallam University

RSA: Implementation

- Alice chooses two ‘large’ primes p and q.

RSA: Implementation

- Alice chooses two 'large' primes p and q.
- She calculates $n = pq$ (the **modulus**) and then finds $\varphi(n) = \varphi(pq) = (p-1)(q-1)$.

RSA: Implementation

- Alice chooses two 'large' primes p and q.
- She calculates $n = pq$ (the **modulus**) and then finds $\varphi(n) = \varphi(pq) = (p-1)(q-1)$.
- Alice also selects a value of e (the **encryption exponent**) such that $1 < e < \varphi(n)$ and $\text{hcf}(e, \varphi(n)) = 1$.

RSA: Implementation

- Alice chooses two ‘large’ primes p and q.
- She calculates $n = pq$ (the **modulus**) and then finds $\varphi(n) = \varphi(pq) = (p-1)(q-1)$.
- Alice also selects a value of e (the **encryption exponent**) such that $1 < e < \varphi(n)$ and $\text{hcf}(e, \varphi(n)) = 1$.
- She also finds a value d (the **decryption exponent**) such that $de = 1 \bmod \varphi(n)$. (More on this later.)

RSA: Implementation

- Alice chooses two ‘large’ primes p and q.
- She calculates $n = pq$ (the **modulus**) and then finds $\varphi(n) = \varphi(pq) = (p-1)(q-1)$.
- Alice also selects a value of e (the **encryption exponent**) such that $1 < e < \varphi(n)$ and $\text{hcf}(e, \varphi(n)) = 1$.
- She also finds a value d (the **decryption exponent**) such that $de = 1 \bmod \varphi(n)$. (More on this later.)
- Alice then publishes (n, e) publicly: this is her **public key**.

RSA: Implementation

- Alice chooses two 'large' primes p and q.
- She calculates $n = pq$ (the **modulus**) and then finds $\varphi(n) = \varphi(pq) = (p-1)(q-1)$.
- Alice also selects a value of e (the **encryption exponent**) such that $1 < e < \varphi(n)$ and $\text{hcf}(e, \varphi(n)) = 1$.
- She also finds a value d (the **decryption exponent**) such that $de = 1 \bmod \varphi(n)$. (More on this later.)
- Alice then publishes (n, e) publicly: this is her **public key**.
- She keeps the value of d secret: this is her **private key**.

RSA: Encryption

- Alice has published her public key (n, e) and kept her private key d secret.

RSA: Encryption

- Alice has published her public key (n, e) and kept her private key d secret.
- Bob can now encrypt messages to send to Alice. He selects a message value $0 \leq m < n$ and calculates $c = m^e \bmod n$. He sends this to Alice.

RSA: Encryption

- Alice has published her public key (n, e) and kept her private key d secret.
- Bob can now encrypt messages to send to Alice. He selects a message value $0 \leq m < n$ and calculates $c = m^e \bmod n$. He sends this to Alice.
- Alice can now decrypt this by finding $c^d = m^{de} = m \bmod n$.

RSA: Encryption

- Alice has published her public key (n, e) and kept her private key d secret.
- Bob can now encrypt messages to send to Alice. He selects a message value $0 \leq m < n$ and calculates $c = m^e \bmod n$. He sends this to Alice.
- Alice can now decrypt this by finding $c^d = m^{de} = m \bmod n$.
- That this works relies on a result in number theory called Euler's Theorem:
  - Let n be a non-negative integer and let a be an integer coprime to n. Then

$$a^{\varphi(n)} = 1 \bmod n.$$

RSA: Encryption Example

- Suppose that Alice has chosen primes $p = 7$ and $q = 13$, and **encryption exponent** $e = 5$.

RSA: Encryption Example

- Suppose that Alice has chosen primes $p = 7$ and $q = 13$, and **encryption exponent** $e = 5$.
- Her **modulus** is $n = pq = 7 \times 13 = 91$.

RSA: Encryption Example

- Suppose that Alice has chosen primes $p = 7$ and $q = 13$, and **encryption exponent** $e = 5$.
- Her **modulus** is $n = pq = 7 \times 13 = 91$.
- She also calculates $\varphi(n) = \varphi(91) = \varphi(7 \times 13) = 6 \times 12 = 72$.

RSA: Encryption Example

- Suppose that Alice has chosen primes $p = 7$ and $q = 13$, and **encryption exponent** $e = 5$.
- Her **modulus** is $n = pq = 7 \times 13 = 91$.
- She also calculates $\varphi(n) = \varphi(91) = \varphi(7 \times 13) = 6 \times 12 = 72$.
- Alice finds $d = 29$, so that $de = 29 \times 5 = 145 = 1 \bmod 72$.

RSA: Encryption Example

- Suppose that Alice has chosen primes $p = 7$ and $q = 13$, and **encryption exponent** $e = 5$.
- Her **modulus** is $n = pq = 7 \times 13 = 91$.
- She also calculates $\varphi(n) = \varphi(91) = \varphi(7 \times 13) = 6 \times 12 = 72$.
- Alice finds $d = 29$, so that $de = 29 \times 5 = 145 = 1 \bmod 72$.
- She publishes the public key $(n, e) = (91, 5)$.

RSA: Encryption Example

- Suppose that Alice has chosen primes $p = 7$ and $q = 13$, and **encryption exponent** $e = 5$.
- Her **modulus** is $n = pq = 7 \times 13 = 91$.
- She also calculates $\varphi(n) = \varphi(91) = \varphi(7 \times 13) = 6 \times 12 = 72$.
- Alice finds $d = 29$, so that $de = 29 \times 5 = 145 = 1 \bmod 72$.
- She publishes the public key $(n, e) = (91, 5)$.
- Bob wants to send the message '`Y`' to Alice. He could convert this to ASCII (0101 1001 in binary, 89 in denary) so that the message value is $m = 89$.

RSA: Encryption Example

- Suppose that Alice has chosen primes $p = 7$ and $q = 13$, and **encryption exponent** $e = 5$.
- Her **modulus** is $n = pq = 7 \times 13 = 91$.
- She also calculates $\varphi(n) = \varphi(91) = \varphi(7 \times 13) = 6 \times 12 = 72$.
- Alice finds $d = 29$, so that $de = 29 \times 5 = 145 = 1 \bmod 72$.
- She publishes the public key $(n, e) = (91, 5)$.
- Bob wants to send the message '`Y`' to Alice. He could convert this to ASCII (0101 1001 in binary, 89 in denary) so that the message value is $m = 89$.
- Bob calculates $c = m^e = 89^5 \bmod 91$. He might do this by repeated squaring if the powers are large. (See previous notes on modular arithmetic.)

RSA: Encryption Example

- Suppose that Alice has chosen primes $p = 7$ and $q = 13$, and **encryption exponent** $e = 5$.
- Her **modulus** is $n = pq = 7 \times 13 = 91$.
- She also calculates $\varphi(n) = \varphi(91) = \varphi(7 \times 13) = 6 \times 12 = 72$.
- Alice finds $d = 29$, so that $de = 29 \times 5 = 145 = 1 \bmod 72$.
- She publishes the public key $(n, e) = (91, 5)$.
- Bob wants to send the message '`Y`' to Alice. He could convert this to ASCII (0101 1001 in binary, 89 in denary) so that the message value is $m = 89$.
- Bob calculates $c = m^e = 89^5 \bmod 91$. He might do this by repeated squaring if the powers are large. (See previous notes on modular arithmetic.)
- $c = m^e = 89^5 = (-2)^5 = -32 = 59 \bmod 91$

RSA: Encryption Example

- Suppose that Alice has chosen primes $p = 7$ and $q = 13$, and **encryption exponent** $e = 5$.
- Her **modulus** is $n = pq = 7 \times 13 = 91$.
- She also calculates $\varphi(n) = \varphi(91) = \varphi(7 \times 13) = 6 \times 12 = 72$.
- Alice finds $d = 29$, so that $de = 29 \times 5 = 145 = 1 \bmod 72$.
- She publishes the public key $(n, e) = (91, 5)$.
- Bob wants to send the message '`Y`' to Alice. He could convert this to ASCII (0101 1001 in binary, 89 in denary) so that the message value is $m = 89$.
- Bob calculates $c = m^e = 89^5 \bmod 91$. He might do this by repeated squaring if the powers are large. (See previous notes on modular arithmetic.)
- $c = m^e = 89^5 = (-2)^5 = -32 = 59 \bmod 91$
- Alice decrypts this by calculating $c^d = 59^{29} \bmod 91$.

Quick Powers: Recap

- To decrypt Bob's encrypted message, Alice calculates $c^d = 59^{29} \bmod 91$.

Quick Powers: Recap

- To decrypt Bob's encrypted message, Alice calculates $c^d = 59^{29} \bmod 91$.
- The power is $29 = 16 + 8 + 4 + 1$. We write it as the sum of powers of 2, then $59^{29} = 59^{16} \times 59^{8} \times 59^{4} \times 59^{1}$.

Quick Powers: Recap

- To decrypt Bob's encrypted message, Alice calculates $c^d = 59^{29} \bmod 91$.
- The power is $29 = 16 + 8 + 4 + 1$. We write it as the sum of powers of 2, then $59^{29} = 59^{16} \times 59^{8} \times 59^{4} \times 59^{1}$.
-

$$
\begin{aligned}
59^1 &= 59 \bmod 91 \\
59^2 &= 3481 = 23 \bmod 91 \\
59^4 &= 23^2 = 529 = 74 \bmod 91 \\
59^8 &= 74^2 = 5476 = 16 \bmod 91 \\
59^{16} &= 16^2 = 256 = 74 \bmod 91
\end{aligned}
$$

Quick Powers: Recap

- To decrypt Bob's encrypted message, Alice calculates $c^d = 59^{29} \bmod 91$.
- The power is $29 = 16 + 8 + 4 + 1$. We write it as the sum of powers of 2, then $59^{29} = 59^{16} \times 59^8 \times 59^4 \times 59^1$.
-

$$\begin{aligned}
59^1 &= 59 \bmod 91 \\
59^2 &= 3481 = 23 \bmod 91 \\
59^4 &= 23^2 = 529 = 74 \bmod 91 \\
59^8 &= 74^2 = 5476 = 16 \bmod 91 \\
59^{16} &= 16^2 = 256 = 74 \bmod 91
\end{aligned}$$

- $59^{29} = 59^{16} \times 59^8 \times 59^4 \times 59^1 = 74 \times 16 \times 74 \times 59 = 89 \bmod 91$.

Quick Powers: Recap

- To decrypt Bob's encrypted message, Alice calculates $c^d = 59^{29} \bmod 91$.
- The power is $29 = 16 + 8 + 4 + 1$. We write it as the sum of powers of 2, then $59^{29} = 59^{16} \times 59^8 \times 59^4 \times 59^1$.
- $$\begin{aligned} 59^1 &= 59 \bmod 91 \\ 59^2 &= 3481 = 23 \bmod 91 \\ 59^4 &= 23^2 = 529 = 74 \bmod 91 \\ 59^8 &= 74^2 = 5476 = 16 \bmod 91 \\ 59^{16} &= 16^2 = 256 = 74 \bmod 91 \end{aligned}$$
- $59^{29} = 59^{16} \times 59^8 \times 59^4 \times 59^1 = 74 \times 16 \times 74 \times 59 = 89 \bmod 91$.
- In practice we encrypt multiple letters at a time in blocks (255 letters), rather than just single letters. This prevents simple frequency analysis style attacks.

RSA: Decryption Exponent and Euclidean Algorithm

When Alice sets up her RSA scheme, she chooses large primes p and q. Using these she calculates $n = pq$ and $\varphi(n) = (p-1)(q-1)$, but she also requires a value e such that $1 < e < \varphi(n)$ with $\mathrm{hcf}(e, \varphi(n)) = 1$ and a value d such that $de = 1 \bmod \varphi(n)$. This can be hard to find by trial and error, so we can use the **Euclidean Algorithm**.

RSA: Decryption Exponent and Euclidean Algorithm

When Alice sets up her RSA scheme, she chooses large primes p and q. Using these she calculates $n = pq$ and $\varphi(n) = (p-1)(q-1)$, but she also requires a value e such that $1 < e < \varphi(n)$ with $\text{hcf}(e, \varphi(n)) = 1$ and a value d such that $de = 1 \bmod \varphi(n)$. This can be hard to find by trial and error, so we can use the **Euclidean Algorithm**.

We can easily check that the highest common factor of $e = 5$ and $\varphi(n) = 72$ is actually equal to 1.

RSA: Decryption Exponent and Euclidean Algorithm

When Alice sets up her RSA scheme, she chooses large primes p and q. Using these she calculates $n = pq$ and $\varphi(n) = (p-1)(q-1)$, but she also requires a value e such that $1 < e < \varphi(n)$ with $\text{hcf}(e, \varphi(n)) = 1$ and a value d such that $de = 1 \bmod \varphi(n)$. This can be hard to find by trial and error, so we can use the **Euclidean Algorithm**.
We can easily check that the highest common factor of $e = 5$ and $\varphi(n) = 72$ is actually equal to 1.

$$\begin{aligned} 72 &= 14 \times 5 + 2 \\ 5 &= 2 \times 2 + 1 \\ 2 &= 2 \times 1 + 0 \end{aligned}$$

RSA: Decryption Exponent and Euclidean Algorithm

When Alice sets up her RSA scheme, she chooses large primes p and q. Using these she calculates $n = pq$ and $\varphi(n) = (p-1)(q-1)$, but she also requires a value e such that $1 < e < \varphi(n)$ with $\text{hcf}(e, \varphi(n)) = 1$ and a value d such that $de = 1 \bmod \varphi(n)$. This can be hard to find by trial and error, so we can use the **Euclidean Algorithm**.
We can easily check that the highest common factor of $e = 5$ and $\varphi(n) = 72$ is actually equal to 1.

$$
\begin{aligned}
72 &= 14 \times 5 + 2 \\
5 &= 2 \times 2 + 1 \\
2 &= 2 \times 1 + 0
\end{aligned}
$$

The least number before 0 is the highest common factor. Here this is 1, so e and $\varphi(n)$ are coprime.

Extended Euclidean Algorithm

The RSA example used the value of $d = 29$, but didn't explain where this came from.

Extended Euclidean Algorithm

The RSA example used the value of $d = 29$, but didn't explain where this came from.

$$
\begin{array}{ccccc}
 & \varphi(n) & & e & \\
(1,0) & 72 & & 5 & (0,1) \\
 & & \times 14 & & \\
(0,14) & 70 & & 4 & (2,-28) \\
\cline{4-5}
 & & \times 2 & & \\
\cline{1-2}
(1,-14) & 2 & & 1 & (-2,29) \\
 & & \times 2 & & \\
(-4,58) & 2 & & & \\
\cline{1-2}
(5,-72) & 0 & & &
\end{array}
$$

Extended Euclidean Algorithm

The RSA example used the value of $d = 29$, but didn't explain where this came from.

$$
\begin{array}{ccccc}
 & \varphi(n) & & e & \\
(1,0) & 72 & & 5 & (0,1) \\
 & & \times 14 & & \\
(0,14) & 70 & & 4 & (2,-28) \\
\cline{4-5}
 & & \times 2 & & \\
\cline{1-2}
(1,-14) & 2 & & 1 & (-2,29) \\
 & & \times 2 & & \\
(-4,58) & 2 & & & \\
\cline{1-2}
(5,-72) & 0 & & &
\end{array}
$$

At each stage, the pair of coordinates (a, b) corresponding to a number c in the table tells us that $72a + 5b = c$. E.g., the pair $(2, -28)$ corresponds to the number 4 in the table, so we know that $72 \times 2 + 5 \times (-28) = 4$.

Extended Euclidean Algorithm

The RSA example used the value of $d = 29$, but didn't explain where this came from.

	$\varphi(n)$		e	
$(1, 0)$	72		5	$(0, 1)$
		$\times 14$		
$(0, 14)$	70		4	$(2, -28)$
		$\times 2$		
$(1, -14)$	2		1	$(-2, 29)$
		$\times 2$		
$(-4, 58)$	2			
$(5, -72)$	0			

At each stage, the pair of coordinates (a, b) corresponding to a number c in the table tells us that $72a + 5b = c$. E.g., the pair $(2, -28)$ corresponds to the number 4 in the table, so we know that $72 \times 2 + 5 \times (-28) = 4$. Similarly, we know that $72 \times (-2) + 5 \times 29 = 1$.

Extended Euclidean Algorithm

The RSA example used the value of $d = 29$, but didn't explain where this came from.

	$\varphi(n)$		e	
$(1,0)$	72		5	$(0,1)$
		$\times 14$		
$(0,14)$	70		4	$(2,-28)$
		$\times 2$		
$(1,-14)$	2		1	$(-2,29)$
		$\times 2$		
$(-4,58)$	2			
$(5,-72)$	0			

At each stage, the pair of coordinates (a,b) corresponding to a number c in the table tells us that $72a + 5b = c$. E.g., the pair $(2,-28)$ corresponds to the number 4 in the table, so we know that $72 \times 2 + 5 \times (-28) = 4$. Similarly, we know that $72 \times (-2) + 5 \times 29 = 1$.

We need a value of d such that $de = 1 \bmod \varphi(n)$, so in our case such that $5d = 1 \bmod 72$. Now we know that $72 \times (-2) + 5 \times 29 = 1$, so $5 \times 29 = 1 \bmod 72$. Hence $d = 29$.

RSA: Example

Alice changes her RSA scheme to use $p = 11$, $q = 19$, and $e = 7$.

RSA: Example

Alice changes her RSA scheme to use $p = 11$, $q = 19$, and $e = 7$. She calculates $n = pq = 11 \times 19 = 209$ and $\varphi(n) = (p-1)(q-1) = 10 \times 18 = 180$. She publishes $(n, e) = (209, 7)$ and calculates d using the extended Euclidean algorithm.

RSA: Example

Alice changes her RSA scheme to use $p = 11$, $q = 19$, and $e = 7$. She calculates $n = pq = 11 \times 19 = 209$ and $\varphi(n) = (p-1)(q-1) = 10 \times 18 = 180$. She publishes $(n, e) = (209, 7)$ and calculates d using the extended Euclidean algorithm.

$$
\begin{array}{ccccc}
 & \varphi(n) & & e & \\
(1,0) & 180 & & 7 & (0,1) \\
 & & \times 25 & & \\
(0,25) & 175 & & 5 & (1,-25) \\
\cline{4-5}
 & & \times 1 & & \\
\cline{1-2}
(1,-25) & 5 & & 2 & (-1,26) \\
 & & \times 2 & & \\
(-2,52) & 4 & & & \\
\cline{1-2}
(3,-77) & 1 & & &
\end{array}
$$

RSA: Example

Alice changes her RSA scheme to use $p = 11$, $q = 19$, and $e = 7$. She calculates $n = pq = 11 \times 19 = 209$ and $\varphi(n) = (p-1)(q-1) = 10 \times 18 = 180$. She publishes $(n, e) = (209, 7)$ and calculates d using the extended Euclidean algorithm.

$$
\begin{array}{cccccc}
 & \varphi(n) & & e & \\
(1,0) & 180 & & 7 & (0,1) \\
 & & \times 25 & & \\
(0,25) & 175 & & 5 & (1,-25) \\
 & & \times 1 & & \\
(1,-25) & 5 & & 2 & (-1,26) \\
 & & \times 2 & & \\
(-2,52) & 4 & & & \\
(3,-77) & 1 & & &
\end{array}
$$

The algorithm suggests that $d = -77$, but we want d to be positive. Since we want $de = 1 \bmod \varphi(n)$ and $\varphi(n) = 180$, we can add 180 to -77 to find that

$$d = -77 = -77 + 180 = 103 \bmod 180.$$

RSA in Practice

- The main idea behind RSA is that your encryption key (n, e) can be made **public**: anybody can know it without any security issues. But, only you have access to the decryption key (n, d).

RSA in Practice

- The main idea behind RSA is that your encryption key (n, e) can be made **public**: anybody can know it without any security issues. But, only you have access to the decryption key (n, d).
- Theoretically, an attacker could factor the modulus n and use the primes to calculate the decryption exponent d. The security of RSA depends on this being a very difficult problem.

RSA in Practice

- The main idea behind RSA is that your encryption key (n, e) can be made **public**: anybody can know it without any security issues. But, only you have access to the decryption key (n, d).
- Theoretically, an attacker could factor the modulus n and use the primes to calculate the decryption exponent d. The security of RSA depends on this being a very difficult problem.
- Our examples only include small primes so that we can understand how RSA works. In practice, the modulus will be 2048 or 3072 bits as recommended by NIST in 2015. This means that the modulus will be of the order of 600+ decimal digits.

RSA in Practice

- The main idea behind RSA is that your encryption key (n, e) can be made **public**: anybody can know it without any security issues. But, only you have access to the decryption key (n, d).
- Theoretically, an attacker could factor the modulus n and use the primes to calculate the decryption exponent d. The security of RSA depends on this being a very difficult problem.
- Our examples only include small primes so that we can understand how RSA works. In practice, the modulus will be 2048 or 3072 bits as recommended by NIST in 2015. This means that the modulus will be of the order of 600+ decimal digits.
- In general, RSA is not recommended for encrypting whole messages as it is computationally impractical. Instead it finds most of its use in key distribution (alongside Diffie-Hellman key exchange) and digital verification/authentication schemes.

RSA Public Key: shu.ac.uk

- shu.ac.uk uses the PKCS #1 RSA Encryption specification.

RSA Public Key: shu.ac.uk

- shu.ac.uk uses the PKCS #1 RSA Encryption specification.
- It uses public key

  ```
  30 82 01 0a 02 82 01 01 00 ad ef f1 f6 86 63 6a 58 60 b7 d9 16 ad ed 3b
  8a 76 f5 47 30 aa a2 b2 b7 14 4c 57 72 c8 8f 11 9b 7d 62 2f 15 15 45 0e
  c8 4c 66 97 00 d1 9e f3 9a 59 9a 03 e3 93 93 a6 38 d9 16 5d ce 1e f6 02
  f1 07 f3 ba 3e f1 a0 97 b6 d5 95 8c 5a c2 ad 88 04 15 86 15 08 45 a9 18
  81 01 dd 75 ba 7e 31 07 78 3a f6 05 93 3e 37 90 fc 99 61 c2 ad 5f 69 a4
  f3 59 b5 b5 90 f0 8e 14 97 be 31 6e c4 75 5b bf eb 11 4a 1f 87 c1 8f 97
  ae ae 80 2f 8c 77 6a f8 c7 5c a2 d9 0b dd 3e 2f e4 03 d0 b4 d5 be 4f 50
  8a d0 f7 c7 ea 7c 8c a5 69 94 7e dd 4b 72 e5 e0 fa 7a f9 be 38 f9 d9 90
  af 34 d5 02 92 a2 99 3c 16 fb 23 e9 2d 0a 74 a1 38 f6 71 e2 8b 7a e1 ff
  0b b1 b1 85 61 8f dc 21 9f e7 81 92 4c f3 24 01 df e1 c7 d2 66 94 f7 9b
  8a 70 25 71 cf db a9 c4 94 a7 b0 e9 69 a6 2f ff a0 a5 bc f6 5f dc 23 6e
  cf 02 03 01 00 01
  ```

- The modulus is the part of the key in black. The encryption exponent is the last three bytes (in green). The rest is to do with Distinguished Encoding Rules.

RSA Public Key: shu.ac.uk

- shu.ac.uk uses the PKCS #1 RSA Encryption specification.
- It uses public key

  ```
  30 82 01 0a 02 82 01 01 00 ad ef f1 f6 86 63 6a 58 60 b7 d9 16 ad ed 3b
  8a 76 f5 47 30 aa a2 b2 b7 14 4c 57 72 c8 8f 11 9b 7d 62 2f 15 15 45 0e
  c8 4c 66 97 00 d1 9e f3 9a 59 9a 03 e3 93 93 a6 38 d9 16 5d ce 1e f6 02
  f1 07 f3 ba 3e f1 a0 97 b6 d5 95 8c 5a c2 ad 88 04 15 86 15 08 45 a9 18
  81 01 dd 75 ba 7e 31 07 78 3a f6 05 93 3e 37 90 fc 99 61 c2 ad 5f 69 a4
  f3 59 b5 b5 90 f0 8e 14 97 be 31 6e c4 75 5b bf eb 11 4a 1f 87 c1 8f 97
  ae ae 80 2f 8c 77 6a f8 c7 5c a2 d9 0b dd 3e 2f e4 03 d0 b4 d5 be 4f 50
  8a d0 f7 c7 ea 7c 8c a5 69 94 7e dd 4b 72 e5 e0 fa 7a f9 be 38 f9 d9 90
  af 34 d5 02 92 a2 99 3c 16 fb 23 e9 2d 0a 74 a1 38 f6 71 e2 8b 7a e1 ff
  0b b1 b1 85 61 8f dc 21 9f e7 81 92 4c f3 24 01 df e1 c7 d2 66 94 f7 9b
  8a 70 25 71 cf db a9 c4 94 a7 b0 e9 69 a6 2f ff a0 a5 bc f6 5f dc 23 6e
  cf 02 03 01 00 01
  ```

- The modulus is the part of the key in black. The encryption exponent is the last three bytes (in green). The rest is to do with Distinguished Encoding Rules.

RSA Public Key: shu.ac.uk

- The actual modulus in denary is:
  1439009874334747912175708580696616782489633913392424042420786858
  8493385513036561327933889300201125505036733509500125430474349113
  8392109631132209102846208024434773766208990866664115880472151837
  2314480955340551600970256190940881256277567983112709841760074703
  7746991422383533253603181640828388090653889268036267798634609551
  9367579321471090171158258338355210007966141574788860342131168459
  2859805329240698372132726428980674548023372409653959742420976883
  5330296670239424922714039292364401817591460156344469804908668602
  9534570294078337741486136447231027678544026989306624260832616420
  656060942401293996193415572491513886591674009 9.

RSA Public Key: shu.ac.uk

- The actual modulus in denary is:
  14390098743347479121757085806966167824896339133924240424207868588
  49338551303656132793388930020112555050367335095001254304743491138
  39210963113220910284620802443477376620899086666411588047215183723
  14480955340551600970256190940881256277567983112709841760074703774
  69914223835332536031816408283880906538892680362677986346095519367
  57932147109017115825833835521000796614157478886034213116845928598
  05329240698372132726428980674548023372409653959742420976883533029
  66702394249227140392923644018175914601563444698049086686029534570
  29407833774148613644723102767854402698930662426083261642065606094
  24012939961934155724915138865916740099.
- The encryption exponent in denary is: 65537.

RSA Public Key: shu.ac.uk

- The actual modulus in denary is:
  1439009874334747912175708580696616782489633913392424042420786858
  8493385513036561327933889300201125505036733509500125430474349113
  8392109631132209102846208024434773766208990866664115880472151837
  2314480955340551600970256190940881256277567983112709841760074703
  7746991422383533253603181640828388090653889268036267798634609551
  9367579321471090171158258338355210007966141574788860342131168459
  2859805329240698372132726428980674548023372409653959742420976883
  5330296670239424922714039292364401817591460156344469804908668602
  9534570294078337741486136447231027678544026989306624260832616420
  6560609424012939961934155724915138865916740099.
- The encryption exponent in denary is: 65537.
- Encryption exponents are often primes of the form $2^n + 1$, as this speeds up computation (repeated squaring). E.g., $2^1 + 1 = 3$, $2^4 + 1 = 17$, or $2^{16} + 1 = 65537$.

RSA in Practice

- Part of PKCS #1 is how to **pad** the message. This is a way of adding random noise to the message to avoid sending the same message in the same way multiple times (certain attacks can exploit this).

RSA in Practice

- Part of PKCS #1 is how to **pad** the message. This is a way of adding random noise to the message to avoid sending the same message in the same way multiple times (certain attacks can exploit this).
- Another common padding scheme is OAEP.

RSA in Practice

- Part of PKCS #1 is how to **pad** the message. This is a way of adding random noise to the message to avoid sending the same message in the same way multiple times (certain attacks can exploit this).
- Another common padding scheme is OAEP.
- Investigating possible attacks on RSA is a very common topic: there are plenty of articles about this in 2024 alone.

Tutorials

In the tutorial this week we will:

- Create a spreadsheet to perform simple RSA encryption and allow us to handle larger modular powers.
- Use the spreadsheet from last week to calculate decryption exponents for RSA.
- Practice implementing RSA.