Cryptography: Public Key Cryptography

Alex Corner

Sheffield Hallam University

Recap: Number Theory

- **Prime number**: a number which is divisible only by itself and by 1, not including 1 itself.

Recap: Number Theory

- **Prime number**: a number which is divisible only by itself and by 1, not including 1 itself.
- An integer a is **coprime** to p if the **highest common factor** of a and p is 1: $\text{hcf}(a, p) = 1$.

Recap: Number Theory

- **Prime number**: a number which is divisible only by itself and by 1, not including 1 itself.
- An integer a is **coprime** to p if the **highest common factor** of a and p is 1: $\text{hcf}(a, p) = 1$. Also known as the **greatest common divisor**.
- Let n be a non-negative integer. The number of integers less than n which are coprime to n is written as $\varphi(n)$. The function φ is known as **Euler's totient function**.

Recap: Number Theory

- **Prime number**: a number which is divisible only by itself and by 1, not including 1 itself.
- An integer a is **coprime** to p if the **highest common factor** of a and p is 1: $\mathrm{hcf}(a, p) = 1$. Also known as the **greatest common divisor**.
- Let n be a non-negative integer. The number of integers less than n which are coprime to n is written as $\varphi(n)$. The function φ is known as **Euler's totient function**.
- This is easy to compute for prime numbers: $\varphi(p) = p - 1$. And for products of primes: $\varphi(pq) = (p-1)(q-1)$.

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 $\mathrm{hcf}(e, \varphi(n)) = 1$.
- 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 $\operatorname{hcf}(e, \varphi(n)) = 1$.
- 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 $\mathrm{hcf}(e, \varphi(n)) = 1$.
- 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 $0 \le m < n$ 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 $\operatorname{hcf}(e, \varphi(n)) = 1$ 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 $\mathrm{hcf}(e, \varphi(n)) = 1$ 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**.
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.

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

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.

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

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 $72 \times 2 + 5 \times (-28) = 4$. 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.

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

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 $72 \times 2 + 5 \times (-28) = 4$. 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.

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

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.

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

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 c6 12 a3 d5 e7 86 86 29 2e 49 28 d8 cb a9 97
  f4 2f ad 0a 11 3a 79 9d c6 d2 61 14 27 9b 86 b4 a2 33 31 58 b6 17 98 b9
  e6 a8 a3 31 1a 3f ac ad e1 ea 0d 10 27 3c 3e b1 f3 e0 2f dc 8c 5e 36 fd
  b6 e9 52 1f 8b 29 4d b6 68 26 99 0d c8 37 77 29 16 28 2b 04 4a af 1c 9d
  65 37 9b a1 9f 45 5f 21 82 df 56 e7 e5 0b ca 9f 6a 74 7b b3 2c 8e f4 f1
  25 75 70 3c e7 a2 ea 0e f8 06 74 36 33 3b 3d 90 a6 a9 e3 0f 36 a3 ed 9e
  96 27 bc 15 99 80 29 ff 54 76 cb df 83 55 a9 e6 bf b9 c6 75 39 d0 95 4f
  d3 a9 3a a1 02 2e 15 6c d6 8d 50 d3 65 cd 8b 9c b1 fb 82 b9 c4 c4 4f 5d
  dd 2a d7 9e 7c 68 90 94 24 5a 96 58 fe e4 b4 42 df 76 ad 53 25 63 13 0c
  93 74 c5 f0 6d 07 cb 4d 4c b6 7c 19 b2 28 35 91 91 9d 58 d5 b2 27 04 96
  73 b2 70 5e 50 02 67 88 e1 de 82 da dc 1f ab 31 af cf 5a df aa 25 25 20
  a1 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 c6 12 a3 d5 e7 86 86 29 2e 49 28 d8 cb a9 97
  f4 2f ad 0a 11 3a 79 9d c6 d2 61 14 27 9b 86 b4 a2 33 31 58 b6 17 98 b9
  e6 a8 a3 31 1a 3f ac ad e1 ea 0d 10 27 3c 3e b1 f3 e0 2f dc 8c 5e 36 fd
  b6 e9 52 1f 8b 29 4d b6 68 26 99 0d c8 37 77 29 16 28 2b 04 4a af 1c 9d
  65 37 9b a1 9f 45 5f 21 82 df 56 e7 e5 0b ca 9f 6a 74 7b b3 2c 8e f4 f1
  25 75 70 3c e7 a2 ea 0e f8 06 74 36 33 3b 3d 90 a6 a9 e3 0f 36 a3 ed 9e
  96 27 bc 15 99 80 29 ff 54 76 cb df 83 55 a9 e6 bf b9 c6 75 39 d0 95 4f
  d3 a9 3a a1 02 2e 15 6c d6 8d 50 d3 65 cd 8b 9c b1 fb 82 b9 c4 c4 4f 5d
  dd 2a d7 9e 7c 68 90 94 24 5a 96 58 fe e4 b4 42 df 76 ad 53 25 63 13 0c
  93 74 c5 f0 6d 07 cb 4d 4c b6 7c 19 b2 28 35 91 91 9d 58 d5 b2 27 04 96
  73 b2 70 5e 50 02 67 88 e1 de 82 da dc 1f ab 31 af cf 5a df aa 25 25 20
  a1 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:
  2500437610131522493100148237076405630918631449591929336512
  2239831454753121949964126846047420776247220203472708577454
  9142809423789955833556783085077726104096884868435708358365
  8982909292834328906237421764600535191834994448584081246697
  6211091677560578034739281193925515725071442095562596056149
  0935466494207680063486570073078223792678356290875731196481
  3634508336439123725107787554955192955326808016969653788768
  8246000841545840648326905294446959029102185819663218454853
  1560100057082391902302619695947540537763672293516289641189
  7411341797564643847884094551642945773927634771284706789180
  9257303725817491140372032614172008609.

RSA Public Key: shu.ac.uk

- The actual modulus in denary is:
  2500437610131522493100148237076405630918631449591929336512
  2239831454753121949964126846047420776247220203472708577454
  9142809423789955833556783085077726104096884868435708358365
  8982909292834328906237421764600535191834994448584081246697
  6211091677560578034739281193925515725071442095562596056149
  0935466494207680063486570073078223792678356290875731196481
  3634508336439123725107787554955192955326808016969653788768
  8246000841545840648326905294446959029102185819663218454853
  1560100057082391902302619695947540537763672293516289641189
  7411341797564643847884094551642945773927634771284706789180
  9257303725817491140372032614172008609.
- The encryption exponent in denary is: 65537.

RSA Public Key: shu.ac.uk

- The actual modulus in denary is:
  2500437610131522493100148237076405630918631449591929336512
  2239831454753121949964126846047420776247220203472708577454
  9142809423789955833556783085077726104096884868435708358365
  8982909292834328906237421764600535191834994448584081246697
  6211091677560578034739281193925515725071442095562596056149
  0935466494207680063486570073078223792678356290875731196481
  3634508336439123725107787554955192955326808016969653788768
  8246000841545840648326905294446959029102185819663218454853
  1560100057082391902302619695947540537763672293516289641189
  7411341797564643847884094551642945773927634771284706789180
  9257303725817491140372032614172008609.
- 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 small examples implementing RSA by hand.