# CSCI 5451 Fall 2015

# Week 11 Notes

Professor Ellen Gethner

October 24, 2015

# Public Key Encryption: The RSA Algorithm

- **Recall** that in public key encryption, our goals for the plaintext message are

# Public Key Encryption: The RSA Algorithm

- **Recall** that in public key encryption, our goals for the plaintext message are

  1. the message must be encrypted,

# Public Key Encryption: The RSA Algorithm

- **Recall** that in public key encryption, our goals for the plaintext message are

    1. the message must be encrypted,
    2. the recipient must be able to read the message,

# Public Key Encryption: The RSA Algorithm

- **Recall** that in public key encryption, our goals for the plaintext message are

  1. the message must be encrypted,
  2. the recipient must be able to read the message,
  3. an evesdropper must not be able to read or compromise the message, and

# Public Key Encryption: The RSA Algorithm

- **Recall** that in public key encryption, our goals for the plaintext message are

  1. the message must be encrypted,
  2. the recipient must be able to read the message,
  3. an evesdropper must not be able to read or compromise the message, and
  4. the recipient must be able to verify that the message is legitimate (*digital signature*).

# Public Key Encryption: RSA

- The idea of public key encryption was invented at Stanford in 1976 by Diffie and Hellman;

# Public Key Encryption: RSA

- The idea of public key encryption was invented at Stanford in 1976 by Diffie and Hellman;
- one resource, among many, is `http://mathworld.wolfram.com/Diffie-HellmanProtocol.html`

# Public Key Encryption: RSA

- The idea of public key encryption was invented at Stanford in 1976 by Diffie and Hellman;
- one resource, among many, is `http://mathworld.wolfram.com/Diffie-HellmanProtocol.html`
- We'll study one particular public key encryption algorithm, namely RSA, which is short for **R**ivest, **S**hamir, and **A**dleman,

# Public Key Encryption: RSA

- The idea of public key encryption was invented at Stanford in 1976 by Diffie and Hellman;
- one resource, among many, is `http://mathworld.wolfram.com/Diffie-HellmanProtocol.html`
- We'll study one particular public key encryption algorithm, namely RSA, which is short for **R**ivest, **S**hamir, and **A**dleman,
- and is one of the best known public key cryptosystems. The main tools are from Number Theory, and

# Public Key Encryption: RSA

- The idea of public key encryption was invented at Stanford in 1976 by Diffie and Hellman;
- one resource, among many, is `http://mathworld.wolfram.com/Diffie-HellmanProtocol.html`
- We'll study one particular public key encryption algorithm, namely RSA, which is short for **R**ivest, **S**hamir, and **A**dleman,
- and is one of the best known public key cryptosystems. The main tools are from Number Theory, and
- in particular, Fermat's Little Theorem is key.

# Public Key Encryption: RSA

- The idea of public key encryption was invented at Stanford in 1976 by Diffie and Hellman;
- one resource, among many, is `http://mathworld.wolfram.com/Diffie-HellmanProtocol.html`
- We'll study one particular public key encryption algorithm, namely RSA, which is short for **R**ivest, **S**hamir, and **A**dleman,
- and is one of the best known public key cryptosystems. The main tools are from Number Theory, and
- in particular, Fermat's Little Theorem is key.
- **Reminder.** Fermat's Little Theorem says that for $a, n \in \mathbb{Z}$ with $n > 1$ and $gcd(a, p) = 1$ we have $a^{p-1} \equiv 1 \pmod{p}$.

# RSA: The Global Procedure

- The basic ingredients of RSA are as follows.

---

[1] hard means that there is no known polynomial-time algorithm to solve the factorization problem

# RSA: The Global Procedure

- The basic ingredients of RSA are as follows.
  - We need two distinct prime numbers $p$ and $q$ and while we're at it, let $n = pq$.

[1]hard means that there is no known polynomial-time algorithm to solve the factorization problem

# RSA: The Global Procedure

- The basic ingredients of RSA are as follows.
  - We need two distinct prime numbers $p$ and $q$ and while we're at it, let $n = pq$.
  - The public encryption key is going to be $n$ together with another number to be announced later.

[1] hard means that there is no known polynomial-time algorithm to solve the factorization problem

# RSA: The Global Procedure

- The basic ingredients of RSA are as follows.
  - We need two distinct prime numbers $p$ and $q$ and while we're at it, let $n = pq$.
  - The public encryption key is going to be $n$ together with another number to be announced later.
  - The private decryption key includes $p$, $q$, and another number to be announced later.

[1] hard means that there is no known polynomial-time algorithm to solve the factorization problem

# RSA: The Global Procedure

- The basic ingredients of RSA are as follows.
  - We need two distinct prime numbers $p$ and $q$ and while we're at it, let $n = pq$.
  - The public encryption key is going to be $n$ together with another number to be announced later.
  - The private decryption key includes $p$, $q$, and another number to be announced later.
  - That $n$ is hard[1] to factor is the trapdoor and the security of the encryption.

---

[1] hard means that there is no known polynomial-time algorithm to solve the factorization problem

# RSA: The Specifics

- First, translate the plaintext message to a numerical message.

# RSA: The Specifics

- First, translate the plaintext message to a numerical message.
- For example,

# RSA: The Specifics

- First, translate the plaintext message to a numerical message.
- For example,
- A = 10, B= 11, C= 12, D= 13, E= 14, F= 15, G= 16, H= 17, I= 18, J= 19, K= 20, L= 21, M= 22, N= 23, O= 24, P= 25, Q= 26, R= 27, S= 28, T= 29, U= 30, V= 31, W= 32, X= 33, Y= 34, Z= 35

# RSA: The Specifics

- First, translate the plaintext message to a numerical message.
- For example,
- A = 10, B= 11, C= 12, D= 13, E= 14, F= 15, G= 16, H= 17, I= 18, J= 19, K= 20, L= 21, M= 22, N= 23, O= 24, P= 25, Q= 26, R= 27, S= 28, T= 29, U= 30, V= 31, W= 32, X= 33, Y= 34, Z= 35
- And we use 99 to represent a blank space.

# RSA: The Specifics

- First, translate the plaintext message to a numerical message.
- For example,
- A = 10, B= 11, C= 12, D= 13, E= 14, F= 15, G= 16, H= 17, I= 18, J= 19, K= 20, L= 21, M= 22, N= 23, O= 24, P= 25, Q= 26, R= 27, S= 28, T= 29, U= 30, V= 31, W= 32, X= 33, Y= 34, Z= 35
- And we use 99 to represent a blank space.
- The message KNOW THYSELF would look like

# RSA: The Specifics

- First, translate the plaintext message to a numerical message.
- For example,
- A $= 10$, B$= 11$, C$= 12$, D$= 13$, E$= 14$, F$= 15$, G$= 16$, H$= 17$, I$= 18$, J$= 19$, K$= 20$, L$= 21$, M$= 22$, N$= 23$, O$= 24$, P$= 25$, Q$= 26$, R$= 27$, S$= 28$, T$= 29$, U$= 30$, V$= 31$, W$= 32$, X$= 33$, Y$= 34$, Z$= 35$
- And we use 99 to represent a blank space.
- The message KNOW THYSELF would look like
- 202324329929173428142115

# RSA specifics, continued

- 202324329929173428142115

# RSA specifics, continued

- 2023243299291734281421115
- The **next step** is to separate the numerical message into blocks of integers, each of which is less than $n = pq$.

# RSA specifics, continued

- 202324329929173428142115

- The **next step** is to separate the numerical message into blocks of integers, each of which is less than $n = pq$.

- In particular, scan 202324329929173428142115 from left to right, separating the digits by stopping just before the new sequence is $> n$.

# RSA. separating the numerical message into blocks

- **For example,** if $p = 149$ and $q = 157$ then $n = 23393$, then 20232432992917342814 2115 becomes

# RSA. separating the numerical message into blocks

- **For example,** if $p = 149$ and $q = 157$ then $n = 23393$, then 202324329929173428142115 becomes
- 20232 4329 9291 7342 8142 115

# RSA. separating the numerical message into blocks

- **For example,** if $p = 149$ and $q = 157$ then $n = 23393$, then 202324329929173428142115 becomes
- 20232 4329 9291 7342 8142 115
- **Prefix Constraint.** Note that making each letter into a two-digit integer avoids ambiguities in the decryption, so there is no problem returning the numerical message back to the plaintext message.

# RSA: The Encryption

- **Encrypting the Message.** We need $n$ and a new integer called $e$.

# RSA: The Encryption

- **Encrypting the Message.** We need $n$ and a new integer called $e$.
- The number $e$ must be able to be inverted $\pmod{\phi(n)}$.

# RSA: The Encryption

- **Encrypting the Message.** We need $n$ and a new integer called $e$.
- The number $e$ must be able to be inverted $\pmod{\phi(n)}$.
- In other words, we need $e$ to satisfy $gcd(e, \phi(n)) = 1$.

# RSA: The Encryption

- **Encrypting the Message.** We need $n$ and a new integer called $e$.
- The number $e$ must be able to be inverted $\pmod{\phi(n)}$.
- In other words, we need $e$ to satisfy $gcd(e, \phi(n)) = 1$.
- If $p$ and $q$ are known, then $\phi(n)$ is easy to compute. Why?

# RSA: The Encryption

- **Encrypting the Message.** We need $n$ and a new integer called $e$.
- The number $e$ must be able to be inverted $\pmod{\phi(n)}$.
- In other words, we need $e$ to satisfy $gcd(e, \phi(n)) = 1$.
- If $p$ and $q$ are known, then $\phi(n)$ is easy to compute. Why?
- **Pause**

# RSA: The Encryption

- **Encrypting the Message.** We need $n$ and a new integer called $e$.
- The number $e$ must be able to be inverted $\pmod{\phi(n)}$.
- In other words, we need $e$ to satisfy $gcd(e, \phi(n)) = 1$.
- If $p$ and $q$ are known, then $\phi(n)$ is easy to compute. Why?
- **Pause**
- $\phi(n) = (p-1)(q-1)$.

# RSA: The Encryption

- **Encrypting the Message.** We need $n$ and a new integer called $e$.
- The number $e$ must be able to be inverted $\pmod{\phi(n)}$.
- In other words, we need $e$ to satisfy $gcd(e, \phi(n)) = 1$.
- If $p$ and $q$ are known, then $\phi(n)$ is easy to compute. Why?
- **Pause**
- $\phi(n) = (p - 1)(q - 1)$.
- **Encryption Key.** The encryption key for the RSA cryptosystem is the pair of integers $(n, e)$.

# RSA: Encrypting the message

- Let $b$ be a block of the message, where $0 \leq b < n$,

# RSA: Encrypting the message

- Let $b$ be a block of the message, where $0 \leq b < n$,
- Denote the encrypted block by $E(b)$. The recipe for computing $E(b)$ is

# RSA: Encrypting the message

- Let $b$ be a block of the message, where $0 \leq b < n$,
- Denote the encrypted block by $E(b)$. The recipe for computing $E(b)$ is
- $E(b)$ is the residue of $b^e \pmod{n}$. That is, reduce $b^e$ modulo $n$.

# RSA: Encrypting the message

- Let $b$ be a block of the message, where $0 \leq b < n$,
- Denote the encrypted block by $E(b)$. The recipe for computing $E(b)$ is
- $E(b)$ is the residue of $b^e \pmod{n}$. That is, reduce $b^e$ modulo $n$.
- Follow the above procedure for each block $b$ of the message and keep the encrypted blocks separate.

# RSA: Encrypting the message

- Let $b$ be a block of the message, where $0 \leq b < n$,
- Denote the encrypted block by $E(b)$. The recipe for computing $E(b)$ is
- $E(b)$ is the residue of $b^e \pmod{n}$. That is, reduce $b^e$ modulo $n$.
- Follow the above procedure for each block $b$ of the message and keep the encrypted blocks separate.
- In our example, for $n = 23393$ we have $\phi(n) = 23088$.

# RSA: Encrypting the message

- Let $b$ be a block of the message, where $0 \leq b < n$,
- Denote the encrypted block by $E(b)$. The recipe for computing $E(b)$ is
- $E(b)$ is the residue of $b^e \pmod{n}$. That is, reduce $b^e$ modulo $n$.
- Follow the above procedure for each block $b$ of the message and keep the encrypted blocks separate.
- In our example, for $n = 23393$ we have $\phi(n) = 23088$.
- Choose $e = 5$ and now compute $E(b)$ for each of the blocks 20232 4329 9291 7342 8142 115.

# RSA: Encrypting the Message

- Below is the *Mathematica* code for the encryption.

# RSA: Encrypting the Message

- Below is the *Mathematica* code for the encryption.

```
In[13]:= theBlocks = {20 232, 4329, 9291, 7342, 8142, 115};
         Table[Mod[theBlocks[[i]]^5, 23 393], {i, 1, Length[theBlocks]}]

Out[14]= {20 036, 23 083, 11 646, 4827, 4446, 13 152}
```

# RSA: Encrypting the Message

- Below is the *Mathematica* code for the encryption.

```
In[13]:= theBlocks = {20 232, 4329, 9291, 7342, 8142, 115};
         Table[Mod[theBlocks[[i]]^5, 23 393], {i, 1, Length[theBlocks]}]
Out[14]= {20 036, 23 083, 11 646, 4827, 4446, 13 152}
```

- In other words, $20232^5 \equiv 20036 \pmod{23393}$ and so on.

# RSA: Encrypting the Message

- Below is the *Mathematica* code for the encryption.

```
In[13]:= theBlocks = {20 232, 4329, 9291, 7342, 8142, 115};
         Table[Mod[theBlocks[[i]] ^ 5, 23 393], {i, 1, Length[theBlocks]}]

Out[14]= {20 036, 23 083, 11 646, 4827, 4446, 13 152}
```

- In other words, $20232^5 \equiv 20036 \pmod{23393}$ and so on.
- One more time with feeling: $E(20232) = 20036$.

# RSA: Encrypting the Message

- Below is the *Mathematica* code for the encryption.

```
In[13]:= theBlocks = {20 232, 4329, 9291, 7342, 8142, 115};
         Table[Mod[theBlocks[[i]]^5, 23 393], {i, 1, Length[theBlocks]}]
Out[14]= {20 036, 23 083, 11 646, 4827, 4446, 13 152}
```

- In other words, $20232^5 \equiv 20036 \pmod{23393}$ and so on.
- One more time with feeling: $E(20232) = 20036$.
- Got it?

# RSA: Decrypting the Message

- How does RSA decryption work?

# RSA: Decrypting the Message

- How does RSA decryption work?
- We need to know

# RSA: Decrypting the Message

- How does RSA decryption work?
- We need to know

  1. $n$, and

# RSA: Decrypting the Message

- How does RSA decryption work?
- We need to know
  1. $n$, and
  2. the inverse of $e \pmod{\phi(n)}$

# RSA: Decrypting the Message

- How does RSA decryption work?
- We need to know
  1. $n$, and
  2. the inverse of $e \pmod{\phi(n)}$
- To compute the inverse of $e \pmod{\phi(n)}$, use the Extended Euclidean Algorithm.

# Extended Euclidean Algorithm and the Multiplicative Inverse of $e \pmod{\phi(n)}$

- In particular, if $a, b \in \mathbb{Z}$ with $gcd(a, b) = D$, then $\exists x, y \in \mathbb{Z}$ such that $ax + by = D$.

# Extended Euclidean Algorithm and the Multiplicative Inverse of $e \pmod{\phi(n)}$

- In particular, if $a, b \in \mathbb{Z}$ with $gcd(a, b) = D$, then $\exists x, y \in \mathbb{Z}$ such that $ax + by = D$.

- In our case, we have $gcd(e, \phi(n)) = 1$ and hence

# Extended Euclidean Algorithm and the Multiplicative Inverse of $e \pmod{\phi(n)}$

- In particular, if $a, b \in \mathbb{Z}$ with $gcd(a, b) = D$, then $\exists x, y \in \mathbb{Z}$ such that $ax + by = D$.
- In our case, we have $gcd(e, \phi(n)) = 1$ and hence
- $\exists d, y \in \mathbb{Z}$ such that $ed - \phi(n)y = 1$

# Extended Euclidean Algorithm and the Multiplicative Inverse of $e \pmod{\phi(n)}$

- In particular, if $a, b \in \mathbb{Z}$ with $gcd(a, b) = D$, then $\exists x, y \in \mathbb{Z}$ such that $ax + by = D$.
- In our case, we have $gcd(e, \phi(n)) = 1$ and hence
- $\exists d, y \in \mathbb{Z}$ such that $ed - \phi(n)y = 1$
- $\Rightarrow ed \equiv 1 \pmod{\phi(n)}$.

# Extended Euclidean Algorithm and the Multiplicative Inverse of $e \pmod{\phi(n)}$

- In particular, if $a, b \in \mathbb{Z}$ with $gcd(a, b) = D$, then $\exists x, y \in \mathbb{Z}$ such that $ax + by = D$.
- In our case, we have $gcd(e, \phi(n)) = 1$ and hence
- $\exists d, y \in \mathbb{Z}$ such that $ed - \phi(n)y = 1$
- $\Rightarrow ed \equiv 1 \pmod{\phi(n)}$.
- **The Point.** We can see that $d$ is the multiplicative inverse of $e \pmod{\phi(n)}$.

# Back to RSA Decryption

- The pair $(n, d)$ is the private key or decryption key.

# Back to RSA Decryption

- The pair $(n, d)$ is the private key or decryption key.
- Let $a$ be a block of the encrypted message and let $D(a)$ denote the decryption of block $a$.

# Back to RSA Decryption

- The pair $(n, d)$ is the private key or decryption key.
- Let $a$ be a block of the encrypted message and let $D(a)$ denote the decryption of block $a$.
- **Goal.** We hope that $D(E(b)) = b$; that is, encryption followed by decryption should yield the original block of the message.

# Back to RSA Decryption

- The pair $(n, d)$ is the private key or decryption key.
- Let $a$ be a block of the encrypted message and let $D(a)$ denote the decryption of block $a$.
- **Goal.** We hope that $D(E(b)) = b$; that is, encryption followed by decryption should yield the original block of the message.
- Thus to prove the correctness of RSA it suffices to show that $D(E(b)) = b$.

# Proof of Correctness of RSA

- **Computations.** Let $b$ be a block of the original message.

# Proof of Correctness of RSA

- **Computations.** Let $b$ be a block of the original message.
- Then $D(E(b)) \equiv D(b^e) \pmod{n}$

# Proof of Correctness of RSA

- **Computations.** Let $b$ be a block of the original message.
- Then $D(E(b)) \equiv D(b^e) \pmod{n}$
- $\equiv (b^e)^d \pmod{n}$

# Proof of Correctness of RSA

- **Computations.** Let $b$ be a block of the original message.
- Then $D(E(b)) \equiv D(b^e) \pmod{n}$
- $\equiv (b^e)^d \pmod{n}$
- $\equiv b^{ed} \pmod{n}$.

# Proof of Correctness of RSA

- **Computations.** Let $b$ be a block of the original message.
- Then $D(E(b)) \equiv D(b^e) \pmod{n}$
- $\equiv (b^e)^d \pmod{n}$
- $\equiv b^{ed} \pmod{n}$.
- **Recall** that $ed = 1 + y\phi(n) = 1 + y(p-1)(q-1)$

# Proof of Correctness of RSA

- **Computations.** Let $b$ be a block of the original message.
- Then $D(E(b)) \equiv D(b^e) \pmod{n}$
- $\equiv (b^e)^d \pmod{n}$
- $\equiv b^{ed} \pmod{n}$.
- **Recall** that $ed = 1 + y\phi(n) = 1 + y(p-1)(q-1)$
- $\Rightarrow b^{ed} = b^{1+y(p-1)(q-1)}$

# Proof of Correctness of RSA

- **Computations.** Let $b$ be a block of the original message.
- Then $D(E(b)) \equiv D(b^e) \pmod{n}$
- $\equiv (b^e)^d \pmod{n}$
- $\equiv b^{ed} \pmod{n}$.
- **Recall** that $ed = 1 + y\phi(n) = 1 + y(p-1)(q-1)$
- $\Rightarrow b^{ed} = b^{1+y(p-1)(q-1)}$
- $\equiv b(b^{p-1})^{y(q-1)}$

# Proof of Correctness of RSA, continued

- So far we have $D(E(b)) \equiv b(b^{p-1})^{y(q-1)}$

# Proof of Correctness of RSA, continued

- So far we have $D(E(b)) \equiv b(b^{p-1})^{y(q-1)}$
- $\equiv b \pmod{p}$. Why?

# Proof of Correctness of RSA, continued

- So far we have $D(E(b)) \equiv b(b^{p-1})^{y(q-1)}$
- $\equiv b \pmod{p}$. Why?
- **Pause**

# Proof of Correctness of RSA, continued

- So far we have $D(E(b)) \equiv b(b^{p-1})^{y(q-1)}$
- $\equiv b \pmod{p}$. Why?
- **Pause**
- There are two cases to consider: either $b = 0$ in which case it is trivial to see that $D(E(b)) = b$ or else

# Proof of Correctness of RSA, continued

- So far we have $D(E(b)) \equiv b(b^{p-1})^{y(q-1)}$
- $\equiv b \pmod{p}$. Why?
- **Pause**
- There are two cases to consider: either $b = 0$ in which case it is trivial to see that $D(E(b)) = b$ or else
- $b \neq 0$ in which case we apply Fermat's Little Theorem to obtain $D(E(b)) = b$, as well.

# Proof of Correctness of RSA, continued

- So far we have $D(E(b)) \equiv b(b^{p-1})^{y(q-1)}$
- $\equiv b \pmod{p}$. Why?
- **Pause**
- There are two cases to consider: either $b = 0$ in which case it is trivial to see that $D(E(b)) = b$ or else
- $b \neq 0$ in which case we apply Fermat's Little Theorem to obtain $D(E(b)) = b$, as well.
- An identical argument (swap the roles of $p$ and $q$) shows that $b^{ed} \equiv b \pmod{q}$.

# Proof of Correctness of RSA, continued

- Here's where we are: $b^{ed} - b \equiv 0 \pmod{p}$ and $b^{ed} - b \equiv 0 \pmod{q}$

# Proof of Correctness of RSA, continued

- Here's where we are: $b^{ed} - b \equiv 0 \pmod{p}$ and $b^{ed} - b \equiv 0 \pmod{q}$
- $\Rightarrow$ both $p$ and $q$ divide $b^{ed} - b$.

# Proof of Correctness of RSA, continued

- Here's where we are: $b^{ed} - b \equiv 0 \pmod{p}$ and $b^{ed} - b \equiv 0 \pmod{q}$
- $\Rightarrow$ both $p$ and $q$ divide $b^{ed} - b$.
- Since $gcd(p, q) = 1$, we have $pq | (b^{ed} - b)$

# Proof of Correctness of RSA, continued

- Here's where we are: $b^{ed} - b \equiv 0 \pmod{p}$ and $b^{ed} - b \equiv 0 \pmod{q}$
- $\Rightarrow$ both $p$ and $q$ divide $b^{ed} - b$.
- Since $gcd(p, q) = 1$, we have $pq | (b^{ed} - b)$
- $\Rightarrow b^{ed} \equiv b \pmod{n}$, as desired.

# Proof of Correctness of RSA, continued

- Here's where we are: $b^{ed} - b \equiv 0 \pmod{p}$ and $b^{ed} - b \equiv 0 \pmod{q}$
- $\Rightarrow$ both $p$ and $q$ divide $b^{ed} - b$.
- Since $gcd(p, q) = 1$, we have $pq | (b^{ed} - b)$
- $\Rightarrow b^{ed} \equiv b \pmod{n}$, as desired.
- **Conclusion.** RSA is correct.

# Proof of Correctness of RSA, continued

- Here's where we are: $b^{ed} - b \equiv 0 \pmod{p}$ and $b^{ed} - b \equiv 0 \pmod{q}$
- $\Rightarrow$ both $p$ and $q$ divide $b^{ed} - b$.
- Since $gcd(p, q) = 1$, we have $pq | (b^{ed} - b)$
- $\Rightarrow b^{ed} \equiv b \pmod{n}$, as desired.
- **Conclusion.** RSA is correct.
- **QED**

# Back to the example

```
In[1]:= theBlocks = {20 232, 4329, 9291, 7342, 8142, 115}
Out[1]= {20 232, 4329, 9291, 7342, 8142, 115}

In[2]:= encryptedBlocks = Table[Mod[theBlocks[[i]]^5, 23 393], {i, 1, Length[theBlocks]}]
Out[2]= {20 036, 23 083, 11 646, 4827, 4446, 13 152}

In[3]:= ExtendedGCD[EulerPhi[23 393], 5]
Out[3]= {1, {2, -9235}}

In[4]:= 2 EulerPhi[23 393] - 9235 × 5
Out[4]= 1

In[5]:= -9235 + EulerPhi[23 393]
Out[5]= 13 853

In[6]:= decryptedBlocks = Table[Mod[encryptedBlocks[[i]]^13 853, 23 393], {i, 1, Length[theBlocks]}]
Out[6]= {20 232, 4329, 9291, 7342, 8142, 115}

In[7]:= % == theBlocks
Out[7]= True
```

# Is RSA secure??

# Question: Why is RSA Secure?

- **Answer.** Suppose you are the evesdropper and you (and everybody else) knows $(n, e)$.

# Question: Why is RSA Secure?

- **Answer.** Suppose you are the evesdropper and you (and everybody else) knows $(n, e)$.

- To find $d$, the decryption key, you need to know $\phi(n)$.

# Question: Why is RSA Secure?

- **Answer.** Suppose you are the evesdropper and you (and everybody else) knows $(n, e)$.
- To find $d$, the decryption key, you need to know $\phi(n)$.
- To compute $\phi(n)$, most people believe that you must be able to factor $n$.

# Question: Why is RSA Secure?

- **Answer.** Suppose you are the evesdropper and you (and everybody else) knows $(n, e)$.

- To find $d$, the decryption key, you need to know $\phi(n)$.

- To compute $\phi(n)$, most people believe that you must be able to factor $n$.

- Factoring an arbitrary integer is a **hard problem**, which means there is no known polynomial-time algorithm for integer factorization.

# Question: Why is RSA Secure?

- **Answer.** Suppose you are the evesdropper and you (and everybody else) knows $(n, e)$.

- To find $d$, the decryption key, you need to know $\phi(n)$.

- To compute $\phi(n)$, most people believe that you must be able to factor $n$.

- Factoring an arbitrary integer is a **hard problem**, which means there is no known polynomial-time algorithm for integer factorization.

- In an upcoming lecture on NP-completeness, we'll quantify the word "hard."

# Question: Why is RSA Secure?

- **Answer.** Suppose you are the evesdropper and you (and everybody else) knows $(n, e)$.
- To find $d$, the decryption key, you need to know $\phi(n)$.
- To compute $\phi(n)$, most people believe that you must be able to factor $n$.
- Factoring an arbitrary integer is a **hard problem**, which means there is no known polynomial-time algorithm for integer factorization.
- In an upcoming lecture on NP-completeness, we'll quantify the word "hard."
- For now, suffice it to say that the security of RSA lies in the fact that it is hard to factor an arbitrary integer.

# But Wait!

- Why not just use brute force to factor an arbitrary integer?

[2] from The Mathematics of Ciphers: Number Theory and RSA Cryptography, by S.C. Coutinho.

# But Wait!

- Why not just use brute force to factor an arbitrary integer?
- **Brief History**[2] In 1977, the inventors of RSA tested their cryptosystem with an encryption key of 129 digits.

[2] from The Mathematics of Ciphers: Number Theory and RSA Cryptography, by S.C. Coutinho.

# But Wait!

- Why not just use brute force to factor an arbitrary integer?
- **Brief History**[2] In 1977, the inventors of RSA tested their cryptosystem with an encryption key of 129 digits.
- At that time with the available algorithms and hardware, factoring a 129 digit number and decrypting the message would have taken 40 quadrillion years.

[2] from The Mathematics of Ciphers: Number Theory and RSA Cryptography, by S.C. Coutinho.

# But Wait!

- Why not just use brute force to factor an arbitrary integer?
- **Brief History**[2] In 1977, the inventors of RSA tested their cryptosystem with an encryption key of 129 digits.
- At that time with the available algorithms and hardware, factoring a 129 digit number and decrypting the message would have taken 40 quadrillion years.
- How big is 40 quadrillion??

[2] from The Mathematics of Ciphers: Number Theory and RSA Cryptography, by S.C. Coutinho.

# But Wait!

- Why not just use brute force to factor an arbitrary integer?
- **Brief History**[2] In 1977, the inventors of RSA tested their cryptosystem with an encryption key of 129 digits.
- At that time with the available algorithms and hardware, factoring a 129 digit number and decrypting the message would have taken 40 quadrillion years.
- How big is 40 quadrillion??
- One quadrillion is $10^{15}$, which means that 40 quadrillion is about $10^{16}$.

[2] from The Mathematics of Ciphers: Number Theory and RSA Cryptography, by S.C. Coutinho.

# But Wait!

- Why not just use brute force to factor an arbitrary integer?
- **Brief History**[2] In 1977, the inventors of RSA tested their cryptosystem with an encryption key of 129 digits.
- At that time with the available algorithms and hardware, factoring a 129 digit number and decrypting the message would have taken 40 quadrillion years.
- How big is 40 quadrillion??
- One quadrillion is $10^{15}$, which means that 40 quadrillion is about $10^{16}$.
- Yikes!

[2] from The Mathematics of Ciphers: Number Theory and RSA Cryptography, by S.C. Coutinho.

## History, continued

- The combination of advances in hardware and new factorization methods and the advent of the internet led to the factorization of the 129 digit RSA challenge key in 1986.

# History, continued

- The combination of advances in hardware and new factorization methods and the advent of the internet led to the factorization of the 129 digit RSA challenge key in 1986.
- It took eight months during which the computers of 600 volunteers in 25 countries were used.

# History, continued

- The combination of advances in hardware and new factorization methods and the advent of the internet led to the factorization of the 129 digit RSA challenge key in 1986.
- It took eight months during which the computers of 600 volunteers in 25 countries were used.
- Each computer worked on a small piece of the problem during idle computer cycles.

# History, continued

- The combination of advances in hardware and new factorization methods and the advent of the internet led to the factorization of the 129 digit RSA challenge key in 1986.
- It took eight months during which the computers of 600 volunteers in 25 countries were used.
- Each computer worked on a small piece of the problem during idle computer cycles.
- All of the pieces of the problem were put together using a supercomputer, yielding the factorization.

# History, continued

- The combination of advances in hardware and new factorization methods and the advent of the internet led to the factorization of the 129 digit RSA challenge key in 1986.
- It took eight months during which the computers of 600 volunteers in 25 countries were used.
- Each computer worked on a small piece of the problem during idle computer cycles.
- All of the pieces of the problem were put together using a supercomputer, yielding the factorization.
- In 1996, RSA-130 was broken in 15% of the time it took to break RSA-129.

- **Moral of the Story.**

- **Moral of the Story.**

- Nothing is secure!

- **Moral of the Story.**
- Nothing is secure!
- New algorithms and technology are constantly pushing the envelope.

# Other Items of Interest Related to RSA

- How should $p$ and $q$ be chosen so as not to compromise the security?

# Other Items of Interest Related to RSA

- How should $p$ and $q$ be chosen so as not to compromise the security?
- In 1995 two university students broke the then current RSA because $p$ and $q$ were poorly chosen.

# Other Items of Interest Related to RSA

- How should $p$ and $q$ be chosen so as not to compromise the security?
- In 1995 two university students broke the then current RSA because $p$ and $q$ were poorly chosen.
- Take number theory in the spring to find out more...

# A Side Trip in Quantum Computing[3]

- **An Astounding Fact.** RSA is **not secure** on a quantum computer.

[3] Chapter 6 in Introduction to Quantum Computers by Berman, Doolen, Mainieri, and Tsifrinovich

# A Side Trip in Quantum Computing[3]

- **An Astounding Fact.** RSA is **not secure** on a quantum computer.
- Anybody have a quantum computer handy?

[3] Chapter 6 in Introduction to Quantum Computers by Berman, Doolen, Mainieri, and Tsifrinovich

# A Side Trip in Quantum Computing[3]

- **An Astounding Fact.** RSA is **not secure** on a quantum computer.
- Anybody have a quantum computer handy?
- Here's why *quantum RSA* is not secure:

[3] Chapter 6 in Introduction to Quantum Computers by Berman, Doolen, Mainieri, and Tsifrinovich

# A Side Trip in Quantum Computing[3]

- **An Astounding Fact.** RSA is **not secure** on a quantum computer.
- Anybody have a quantum computer handy?
- Here's why *quantum RSA* is not secure:
- We have $N$, a product of two primes, and we want to factor it.

[3] Chapter 6 in Introduction to Quantum Computers by Berman, Doolen, Mainieri, and Tsifrinovich

# A Side Trip in Quantum Computing[3]

- **An Astounding Fact.** RSA is **not secure** on a quantum computer.
- Anybody have a quantum computer handy?
- Here's why *quantum RSA* is not secure:
- We have $N$, a product of two primes, and we want to factor it.
- Alternatively, we want to find a proper divisor of $N$ (i.e., not 1 or $N$).

[3] Chapter 6 in Introduction to Quantum Computers by Berman, Doolen, Mainieri, and Tsifrinovich

# A Side Trip in Quantum Computing[3]

- **An Astounding Fact.** RSA is **not secure** on a quantum computer.
- Anybody have a quantum computer handy?
- Here's why *quantum RSA* is not secure:
- We have $N$, a product of two primes, and we want to factor it.
- Alternatively, we want to find a proper divisor of $N$ (i.e., not 1 or $N$).
- Suppose by magic we have an $x \in \mathbb{Z}$ such that $x^2 \equiv 1 \pmod{N}$ with $1 < x < N - 1$.

[3] Chapter 6 in Introduction to Quantum Computers by Berman, Doolen, Mainieri, and Tsifrinovich

# The Insecurity of Quantum RSA

- **So far:** $x \in \mathbb{Z}$ such that $x^2 \equiv 1 \pmod{N}$ with $1 < x < N - 1$

# The Insecurity of Quantum RSA

- **So far:** $x \in \mathbb{Z}$ such that $x^2 \equiv 1 \pmod{N}$ with $1 < x < N - 1$
- $\Rightarrow x^2 - 1 \equiv 0 \pmod{N}$

# The Insecurity of Quantum RSA

- **So far:** $x \in \mathbb{Z}$ such that $x^2 \equiv 1 \pmod{N}$ with $1 < x < N - 1$
- $\Rightarrow x^2 - 1 \equiv 0 \pmod{N}$
- $\Rightarrow x^2 - 1 = kN$ for some $k \in \mathbb{N}$

# The Insecurity of Quantum RSA

- **So far:** $x \in \mathbb{Z}$ such that $x^2 \equiv 1 \pmod{N}$ with $1 < x < N - 1$
- $\Rightarrow x^2 - 1 \equiv 0 \pmod{N}$
- $\Rightarrow x^2 - 1 = kN$ for some $k \in \mathbb{N}$
- $\Rightarrow (x - 1)(x + 1) = kN$

## The Insecurity of Quantum RSA

- **So far:** $x \in \mathbb{Z}$ such that $x^2 \equiv 1 \pmod{N}$ with $1 < x < N - 1$
- $\Rightarrow x^2 - 1 \equiv 0 \pmod{N}$
- $\Rightarrow x^2 - 1 = kN$ for some $k \in \mathbb{N}$
- $\Rightarrow (x - 1)(x + 1) = kN$
- $N | (x - 1)(x + 1)$.

# The Insecurity of Quantum RSA, continued

- That is, $N$ divides $(x-1)(x+1)$ or, alternatively,

# The Insecurity of Quantum RSA, continued

- That is, $N$ divides $(x-1)(x+1)$ or, alternatively,
- we see that $N$ has a nontrivial factor in common with one of $x-1$ or $x+1$ (because $x-1 < x+1 < N$).

# The Insecurity of Quantum RSA, continued

- That is, $N$ divides $(x-1)(x+1)$ or, alternatively,
- we see that $N$ has a nontrivial factor in common with one of $x-1$ or $x+1$ (because $x-1 < x+1 < N$).
- Put another way, $gcd(N, x-1) > 1$ or $gcd(N, x+1) > 1$.

# The Insecurity of Quantum RSA, continued

- That is, $N$ divides $(x-1)(x+1)$ or, alternatively,
- we see that $N$ has a nontrivial factor in common with one of $x-1$ or $x+1$ (because $x-1 < x+1 < N$).
- Put another way, $gcd(N, x-1) > 1$ or $gcd(N, x+1) > 1$.
- In the specific case that $N = pq$, where $p$ and $q$ are distinct primes, it suffices to find one of $p$ or $q$ to factor $N$.

# The Insecurity of Quantum RSA, continued

- That is, $N$ divides $(x-1)(x+1)$ or, alternatively,
- we see that $N$ has a nontrivial factor in common with one of $x-1$ or $x+1$ (because $x-1 < x+1 < N$).
- Put another way, $gcd(N, x-1) > 1$ or $gcd(N, x+1) > 1$.
- In the specific case that $N = pq$, where $p$ and $q$ are distinct primes, it suffices to find one of $p$ or $q$ to factor $N$.
- **Reminder.** The Euclidean Algorithm is fast!

# The Insecurity of Quantum RSA, continued

- That is, $N$ divides $(x-1)(x+1)$ or, alternatively,
- we see that $N$ has a nontrivial factor in common with one of $x-1$ or $x+1$ (because $x-1 < x+1 < N$).
- Put another way, $gcd(N, x-1) > 1$ or $gcd(N, x+1) > 1$.
- In the specific case that $N = pq$, where $p$ and $q$ are distinct primes, it suffices to find one of $p$ or $q$ to factor $N$.
- **Reminder.** The Euclidean Algorithm is fast!
- Thus it suffices to find the magical value of $x$ that satisfies $x^2 \equiv 1 \pmod{N}$.

# The Quantum Way to find the Magical $x$

- First find any $y$ such that $gcd(y, N) = 1$.

[4] Peter Shor, 1994, *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science, IEEE*

# The Quantum Way to find the Magical $x$

- First find any $y$ such that $gcd(y, N) = 1$.
- Thus Euler's formula applies and we have $y^{\phi(N)} \equiv 1 \pmod{N}$.

[4] Peter Shor, 1994, *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science, IEEE*

# The Quantum Way to find the Magical $x$

- First find any $y$ such that $gcd(y, N) = 1$.
- Thus Euler's formula applies and we have $y^{\phi(N)} \equiv 1 \pmod{N}$.
- **Definition.** The **order** of $y \pmod{N}$ is the smallest positive integer $T$ such that $y^T \equiv 1 \pmod{N}$.

[4] Peter Shor, 1994, *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science, IEEE*

# The Quantum Way to find the Magical $x$

- First find any $y$ such that $gcd(y, N) = 1$.
- Thus Euler's formula applies and we have $y^{\phi(N)} \equiv 1 \pmod{N}$.
- **Definition.** The **order** of $y \pmod{N}$ is the smallest positive integer $T$ such that $y^T \equiv 1 \pmod{N}$.
- Euler's formula tells us that such a $T$ exists, although it might be the case that $T < \phi(N)$.

[4] Peter Shor, 1994, *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science, IEEE*

# The Quantum Way to find the Magical $x$

- First find any $y$ such that $gcd(y, N) = 1$.
- Thus Euler's formula applies and we have $y^{\phi(N)} \equiv 1 \pmod{N}$.
- **Definition.** The **order** of $y \pmod{N}$ is the smallest positive integer $T$ such that $y^T \equiv 1 \pmod{N}$.
- Euler's formula tells us that such a $T$ exists, although it might be the case that $T < \phi(N)$.
- **Fact.**[4] There is a quantum algorithm that finds the order of $y$ in $O(L^3)$-time, where $N$ is an $L$-bit number.

[4]Peter Shor, 1994, *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science, IEEE*

# Historical Aside on Quantum Computing

- Shor's Algorithm was the first efficient quantum algorithm that separated classical computing from quantum computing;

# Historical Aside on Quantum Computing

- Shor's Algorithm was the first efficient quantum algorithm that separated classical computing from quantum computing;
- once discovered, it piqued the interest of mathematicians, physicists, and computer scientists (among others) who were interested in the efficacy of quantum computing.

# Historical Aside on Quantum Computing

- Shor's Algorithm was the first efficient quantum algorithm that separated classical computing from quantum computing;
- once discovered, it piqued the interest of mathematicians, physicists, and computer scientists (among others) who were interested in the efficacy of quantum computing.
- It is the ability to efficiently find the order of $y$ that is the key to the breakage of (quantum) RSA.

# Back to the quantum RSA computation

- Recall that $T$ is the order of $y$ so that $y^T \equiv 1 \pmod{N}$ (*) and $T$ is the smallest positive integer $>1$ for which (*) is satisfied.

## Back to the quantum RSA computation

- Recall that $T$ is the order of $y$ so that $y^T \equiv 1 \pmod{N}$ (*) and $T$ is the smallest positive integer $>1$ for which (*) is satisfied.
- Suppose $T$ is even (if not, choose another $y$).

# Back to the quantum RSA computation

- Recall that $T$ is the order of $y$ so that $y^T \equiv 1 \pmod{N}$ (*) and $T$ is the smallest positive integer $>1$ for which (*) is satisfied.

- Suppose $T$ is even (if not, choose another $y$).

- In practice, $y$ is chosen at random and $T$, its order, will be even with probability $\frac{1}{2}$.

# Back to the quantum RSA computation

- Recall that $T$ is the order of $y$ so that $y^T \equiv 1 \pmod{N}$ (*) and $T$ is the smallest positive integer >1 for which (*) is satisfied.
- Suppose $T$ is even (if not, choose another $y$).
- In practice, $y$ is chosen at random and $T$, its order, will be even with probability $\frac{1}{2}$.
- Now let $x = y^{\frac{T}{2}}$.
- Then $x^2 = (y^{\frac{T}{2}})^2 = y^T \equiv 1 \pmod{N}$.

# Back to the quantum RSA computation

- Recall that $T$ is the order of $y$ so that $y^T \equiv 1 \pmod{N}$ (*) and $T$ is the smallest positive integer $>1$ for which (*) is satisfied.
- Suppose $T$ is even (if not, choose another $y$).
- In practice, $y$ is chosen at random and $T$, its order, will be even with probability $\frac{1}{2}$.
- Now let $x = y^{\frac{T}{2}}$.
- Then $x^2 = (y^{\frac{T}{2}})^2 = y^T \equiv 1 \pmod{N}$.
- That is (applause applause) $x^2 \equiv 1 \pmod{N}$ and that is exactly the $x$ we were looking for to break RSA!

# Quantum RSA is not Secure

- **Conclusion.** RSA can be broken in polynomial time on a quantum computer.

# Quantum RSA is not Secure

- **Conclusion.** RSA can be broken in polynomial time on a quantum computer.
- The experts say " RSA is secure on a classical computer, but not on a quantum computer."

**Next Week: Fast Fourier Transform**