FINE 434: FinTech

Lecture 11

Professor Fahad Saleh

McGill University - Desautels

Task: Forge an RSA signature

Given an RSA public key, (e, N), we want to find the secret key, (d, N). We know $e \cdot d \equiv 1 \ (mod \ \phi)$ with $\phi = (p-1)(q-1)$ and p, q being primes such that $N = p \times q$.

Task: Forge an RSA signature

Given an RSA public key, (e, N), we want to find the secret key, (d, N). We know $e \cdot d \equiv 1 \ (mod \ \phi)$ with $\phi = (p-1)(q-1)$ and p, q being primes such that $N = p \times q$.

Therefore, we need to execute two steps to determine d and thus the secret key.

- Find p and q
- Compute the multiplicative inverse of e modulo $\phi = (p-1)(q-1)$

Let's start with the second task...

Finding d from e and ϕ

Recall that $GCD(e, \phi) = 1$ by construction.

Then,

$e \cdot d \equiv 1 \ (mod \ \phi)$

$\Leftrightarrow ed - 1 = k\phi$ for some integer k

$\Leftrightarrow ed - k\phi = 1$ for some integer k

$\Leftrightarrow ed - k\phi = GCD(e, \phi)$ for some integer k

Euclidean Algorithm

Given $a > b > 0$ with $a, b \in \mathbb{N}$ then if $a \,\%\, b == 0$ return b. Otherwise, repeat the algorithm with $a = b$ and $b = a \,\%\, b$.

```
def Euclid(a,b):
    if a%b == 0:
        return b
    else:
        return Euclid(b,a%b)
```

This algorithm returns $GCD(a, b)$. To see that fact, note that $GCD(a, b) = b$ if $a \,\%\, b == 0$ and $GCD(a, b) = GCD(b, a \,\%\, b)$ otherwise.

Extended Euclidean Algorithm

We reverse engineer Euclid's algorithm to find d. Let $a = \phi(N)$ and $b = e$. The final remainder equals the $GCD(e, \phi) = 1$ and all subsequent remainders are linear combinations of e and ϕ.

Example: $e = 3$, $\phi = 8$

$8\%3 = 2$ so $1 * 8 = 2 * 3 + 2$
$3\%2 = 1$ so $1 * 3 = 1 * 2 + 1$
$2\%1 = 0$, so GCD is 1 and the remainder on previous line

$1 * 3 = 1 * 2 + 1$ and $1 * 8 = 2 * 3 + 2 \Leftrightarrow 2 = 1 * 8 - 2 * 3$ so that
$1 * 3 = 1 * (1 * 8 - 2 * 3) + 1 \Leftrightarrow 3 * 3 - 1 * 8 = 1 \Leftrightarrow 3 \cdot e \equiv 1 \ (mod \ \phi)$

$\Longrightarrow \ d = 3$

Extended Euclidean Algorithm Code

```python
def extendedEuclid(a,b,c1=1,c2=0,c3=0,c4=1):
    if a%b == 0:
        return(c3,c4)
    else:
        return(extendedEuclid(b,a%b,c3,c4,c1-c3*int(a/b),c2-c4*int(a/b)))

result = extendedEuclid(8,3)

print("d = "+str(result[1]))
```

```
d = 3
```

At every step: $a = c1 * \phi + c2 * e$ and $b = c3 * \phi + c4 * e$

$$
\begin{aligned}
& a\%b \\
&= a - int(\tfrac{a}{b}) * b \\
&= (c1 * \phi + c2 * e) - int(\tfrac{a}{b}) * c3 * \phi + c4 * e \\
&= (c1 - c3 * int(\tfrac{a}{b})) * \phi + (c2 - c4 * int(\tfrac{a}{b})) * e
\end{aligned}
$$

Alternative: Exhaustive Enumeration

```
1  def modInverse(e, phi):
2      for x in range(1,phi):
3          if (e*x) % phi == 1:
4              return(x)
5      return(None)
6
7  print(modInverse(3,8))
```

```
3
```

Efficiency

The extended Euclidean algorithm finds the modular inverse of e with respect to ϕ in less than one second when both are large ($>$ 512 bits).

In contrast, exhaustive enumeration takes over a second when dealing with much smaller numbers ($<$ 20 bits).

The extended Euclidean algorithm's efficiency means computing d from e and ϕ is not a practical constraint. This means that ϕ must be difficult to compute from N for security. Because $N = p \times q$ with p, q prime and $\phi = (p-1)(q-1)$, RSA security requires that N must be difficult to prime factor.

Computing ϕ

Recall that $\phi = (p-1)(q-1)$ when $N = p \times q$

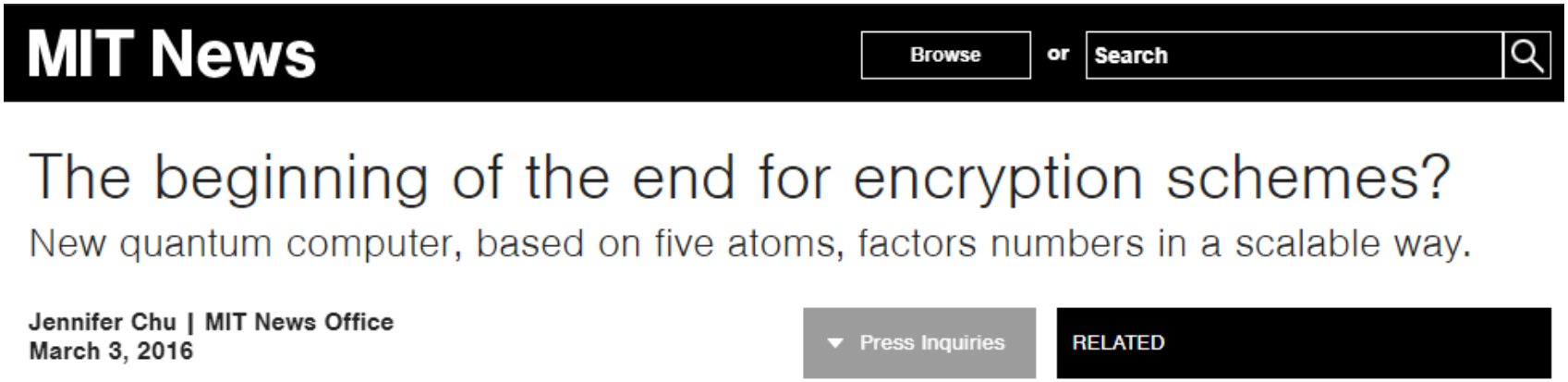
"When the numbers are sufficiently large, no efficient, non-quantum integer factorization algorithm is known. An effort by several researchers, concluded in 2009, to factor a 232-digit number (RSA-768) utilizing hundreds of machines took two years and the researchers estimated that a 1024-bit RSA modulus would take about a thousand times as long."

Semiprimes

“The hardest instances of these problems (for currently known techniques) are semiprimes, the product of two prime numbers.”

“Many cryptographic protocols are based on the difficulty of factoring large composite integers or a related problem—for example, the RSA problem. An algorithm that efficiently factors an arbitrary integer would render RSA-based public-key cryptography insecure.”

Quantum Leap



"It might still cost an enormous amount of money to build — you won't be building a quantum computer and putting it on your desktop anytime soon — but now it's much more an engineering effort, and not a basic physics question."
- Professor Isaac Chuang (MIT)