RSA fast encryption with short public exponent

- RSA ops with public exponent *e* can be speeded-up
  - Encryption, Digital signature verification
  - The public key e can be chosen to be a very small value and RSA is still secure
  - Values
    - e = 3 #MUL + #SQ = 2 (commonly used in practice)
    - e = 17 #MUL + #SQ = 5
    - e = 2^{16}+1 #MUL + #SQ = 17 (avoid small exp attack)
  - gcd(e, Φ(n)) = 1 must hold.

8

Apr-25 The RSA Cryptosystem 61

61

RSA encryption overhead: exercise [→]

- Exercise
  - Assume a 2048-bit modulus and a 32-bit CPU
  - Determine decryption computing overhead (exponentiation)

Assuming half of bits are 1, # multiplications is ≈ 1.5k =
= 2048+1024 = 3072 multiplications
and those are 2048 bits multiplications! So we have to split them in blocks of 32 bits. Result in principle would be 2·2048 bits, so we take modulo n of the multiplication.

Result = a×b mod n

Apr-25 The RSA Cryptosystem 63

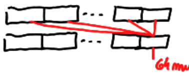
63

RSA encryption overhead: exercise [→]

- Solution (naïve multiplication)
  - On average #MUL+#SQ = 1.5 × 2048 = 3072 *long multiplications* each of which involves 2018-bit operands
  - Cost of a single long-number multiplication
    - Each operand requires 2048/32 = 64 x 32-bit words
    - Each long-number multiplication requires 64^2 = 4096 integer multiplications (integer multiplication is 32 bits × 32 bits)*
    - Modulo reduction requires 64^2 = 4096 integer multiplications
    - In total 4096 + 4096 = 8192 integer multiplications for a single long multiplication
  - Total number of integer multiplications
    - 3072 × 8192 = 25.165.824 integer multiplications

Apr-25 The RSA Cryptosystem 64

64 * In a 32 bits instr. set we can perform 32 bit multiplication.
Big overhead is to make all rows × columns multiplications:



So 64^2 32 bits multiplications. The remainder is of an integer division, complexity is squared again. So 64^2 additional integer multiplications.
So 8192 integer multiplications for a single long multiplication.
Last step: 3072 long multiplications so 1

2

RSA encryption overhead: exercise

- Solution (Montgomery multiplication) Makes multipl. and modulus more efficient
  - Montgomery Multiplication interleaves multiplication and reduction so
  - Number of integer multiplications for one 64×32-bit Montgomery multiplication = **4096**
  - Total number of integer multiplications = **3072** × **4096** = **12.582.912**

Mix the mult. and modulus and be able to do op. in half the operations

- Intuition
  - At each step, the Montgomery method avoids the slow division that naive multiplication leans on, *cutting the work nearly in half*

Apr-25 The RSA Cryptosystem 65

65

(2) This is why public key encryption is computationally demanding. Compared to AES, it's a lot more. That's why you have 2-3 orders of magnitude slower.

RSA encryption

- '70s-'80s: only hardware implementation
- Today, an RSA decryption takes ≈100 μs on high-speed hw
- End '80s, software implementation becomes possible
- Today, 2048-bit RSA takes ≈10 ms on a 2 GHz CPU
  - Throughput = $2048/(10\times10^{-3})$ = 2048×100 = 204.800 bit/s
  - ≈ 3 orders of magnitude slower than symmetric encryption

RSA Fast decryption

- There is no easy way to accelerate RSA when the private exponent *d* is involved
  - sizeof(d) = sizeof(n) to discourage brute force attack
  - It can be shown that sizeof(d) ≥ 0.3 sizeof(n)
- One possible approach is based on the Chinese Remainder Theorem (CRT)
  - We do not prove the theorem
  - We just apply it