

Modular exponentiation

To make RSA practical we must compute $x^D \pmod N$ when N and D are huge.

Example: $N = 101$, $a = 2$, $D = 43$.

$$2^{43} \pmod{101}$$

$$43 = 32 + 11 = 32 + 8 + 3 = 32 + 8 + 2 + 1 = 2^5 + 2^3 + 2^1 + 2^0$$

$$a^{43} = a^{(2^0 + 2^1 + 2^3 + 2^5)} = a^{2^0} \cdot a^{2^1} \cdot a^{2^3} \cdot a^{2^5}$$

$$a = 2 \quad 2^{2^5}$$

$$2^2 = 4 \quad (2^2)^2 = 16 \cdot 2 = 32$$

$$a^{2^n} = (((a^2)^2)^2) \dots$$

$$2, 2^2, 2^4, 2^8, 2^{16}, 2^{32} \quad 2 \cdot (2^2) \cdot (8) \cdot (32)$$

9 mod.

Repeated squaring

Algorithm: Given a , E , and N , the goal is to compute $a^E \pmod{N}$.

- Initialize: Set $S=a$, $P=1$, $T=E$
- Loop: While $T > 1$:
 - if T is odd, set $P = (S * P) \pmod{N}$.
 - Replace S by $S * S \pmod{N}$.
 - Divide T by 2, dropping any remainder
- • Finish: Return $(S * P) \pmod{N}$

S	P	T
<u>2</u>	<u>1</u>	<u>43</u>
4	2	21
16	8	10
54	8	5
88	28	2
68	28	1

16^2

$$16^2 = 256$$

$$\begin{array}{r} 256 \\ -202 \\ \hline 54 \end{array}$$

$\sim \log_2 E$ steps

Return = $86 = 68 * 28 \pmod{101}$.

$$2^{43} \equiv 86 \pmod{101}$$

Proof of repeated squaring

- Initialize: Set $S=a$, $P=1$, $T=E$
 - Loop: While $T>1$:
 - * if T is odd, set $P = (S*P) \bmod N$.
 - * Replace S by $S*S \bmod N$.
 - * Divide T by 2, dropping any remainder
 - Finish: Return $(S*P) \bmod N$

Suppose M is the output of the algorithm with input $S = \underline{a}$, $P = 1$, and $T = E$.

- The first observation we make is that if we initialize the algorithm with $P = k$ instead of $P = 1$, then the output is kM .

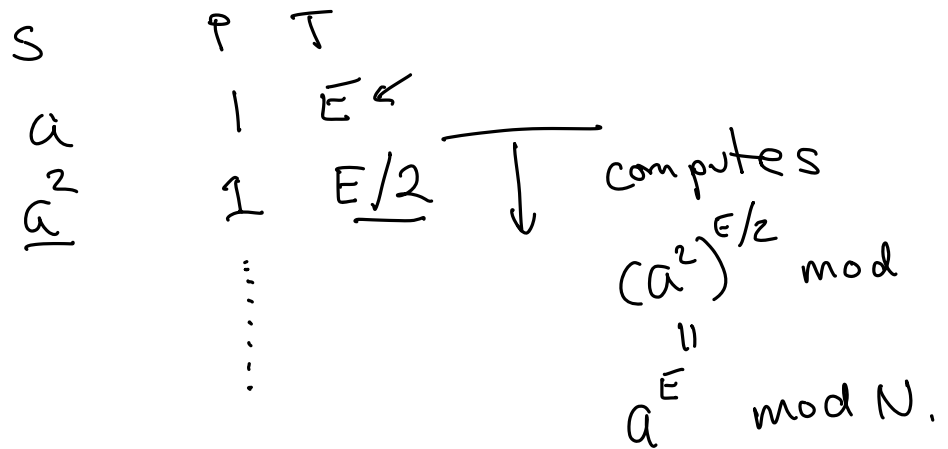
- If $E = 1$, the output is a , as it should be.

$$\begin{aligned} \text{Return } S \cdot P = a^1 & \quad \checkmark \quad S = a \quad P = 1 \\ \text{Return } a & \equiv a^1 \end{aligned}$$

Suppose the algorithm works for all $e < E$, and any a and N . We show it works for E .

If E is even:

- the first step replaces a by a^2 , keeps P as 1 and replaces E by $E/2$.
- By induction the rest of the algorithm computes $(a^2)^{E/2} \bmod N$ or $a^E \bmod N$ as desired.



If E is odd:

- the first step replaces a by a^2 , replaces P by a , and replaces E by $(E - 1)/2$.
- By induction the rest of the algorithm with an initial P of 1 would compute $(a^2)^{(E-1)/2}$ which is $a^{E-1} \pmod N$. But since P started at a , it computes $a a^{E-1} = a^E \pmod N$ as desired.

S	P	T	
a	1	E	
a^2	a	$(E-1)/2$	\downarrow
			$a [a^{(E-1)/2} \pmod N]$
			\downarrow
			\downarrow

$$a \cdot a^{E-1} \pmod N \equiv a^E \pmod N$$

$$\text{Running time} \sim \log_2 E < \log_2 N$$