ECEN 4133

Side channel attacks and defenses

Side channel

- Measure something secret using other available ***indirect measurement***
- Secrets:
  - Passwords
  - Private keys
  - Confidential information
- Available data:
  - Timing
  - Power
  - Heat
  - Sound
  - Pizza deliveries…???
    - Panama invasion (1990)
    - Operation Desert Storm (1991)

Side channel example: passwords

```
bool check_password(char *pw, char *correct) {
    if (strlen(pw)!=strlen(correct))
        return false;
    for (int i=0; i<strlen(pw); i++) {
        if (pw[i] != correct[i]) return false;
    }
    return true;
}
```

Side channel example: passwords

```
bool check_password(char *pw, char *correct) {
        if (strlen(pw)!=strlen(correct))
                return false;
        for (int i=0; i<strlen(pw); i++) {
                if (pw[i] != correct[i]) return false;
        }
        return true;
}
```



```
check_password(“aaa”, “s3cr37”) => 10us
check_password(“aaaaaa”, “s3cr37”) => 15us
check_password(“baaaaa”, “s3cr37”) => 15us
check_password(“saaaaa”, “s3cr37”) => 20us
```

Side channel example: passwords

```
bool check_password(char *pw, char *correct) {
        if (strlen(pw)!=strlen(correct))
                return false;
        for (int i=0; i<strlen(pw); i++) {
                if (pw[i] != correct[i]) return false;
        }
        return true;
}
```



```
check_password(“saaaaa”, “s3cr37”) => 20us
check_password(“s3aaaa”, “s3cr37”) => 25us
```

How many guesses to get correct N-character password?

Side channel example: passwords

- How should we fix this vulnerability?

Side channel solution: constant time

```
// Note: strlen(correct) must be equal to strlen(pw)
// This function still leaks the length of strlen(correct)!
//              (how could we fix?)
bool check_password(char *pw, char *correct) {
        if (strlen(pw) != strlen(correct)) return false;
        int diff = 0;
        for (int i=0; i<strlen(pw); i++) {
                diff |= (pw[i] ^ correct[i]);
        }
        return (diff == 0);
}
```

- check_password(“XXXXXXXXXXX”, “longpassword”) and check_password(“longpassworX”, “longpassword”) should take the same amount of time!

Side channel example: repeated squaring

- Recall RSA decryption/signatures:
  - sig = x^d mod N
  - Where d is a very large number (~2048 bits)
  - Can't write a for loop for this: for (i=0; i<d; i++) { } … never completes (or takes 10^600 years)
  - So how do we compute x^d mod N?

Side channel example: repeated squaring

- Recall RSA decryption/signatures:
  - sig = x^d mod N
  - Where d is a very large number (~2048 bits)
  - Can't write a for loop for this: for (i=0; i<d; i++) { } … never completes (or takes 10^600 years)
  - So how do we compute x^d mod N?
    - Observe that:
      - x^10 == (x^2)^5 == (x^2)*(x^2)^4 == (x^2)*((x^2)^2)^2
      - Similarly: x^256 = (((((((x²)²)²)²)²)²)²)²)²)²)
        x^257 = (((((((x²)²)²)²)²)²)²)²)²)*x
      - x^d should only take log(d) multiplications and/or squaring!

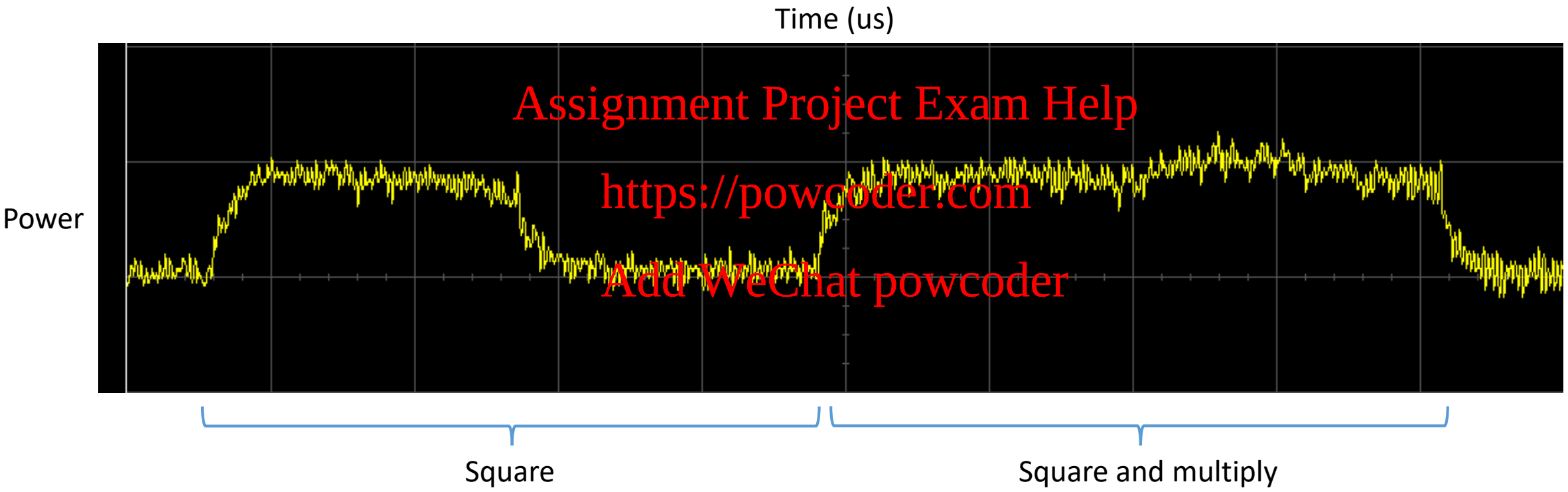
$$x^n = \begin{cases} x\left(x^2\right)^{\frac{n-1}{2}}, & \text{if } n \text{ is odd} \\ \left(x^2\right)^{\frac{n}{2}}, & \text{if } n \text{ is even.} \end{cases}$$

Side channel example: repeated squaring

```
def exp(x, n):
      if n = 0 then return 1
      y = 1;
      while n > 1:
            if n%2==1:  # n is odd
                  y = x * y;
                  x = x * x;
                  n = (n - 1) / 2;
            else:       # n is even
                  x = x * x;
                  n = n / 2;
      return x * y
```


$$x^n = \begin{cases} x\,(x^2)^{\frac{n-1}{2}}, & \text{if } n \text{ is odd} \\ (x^2)^{\frac{n}{2}}, & \text{if } n \text{ is even.} \end{cases}$$

Side channel example: repeated squaring



- Solving the repeated squaring side channel?

Repeated squaring: Montgomery's Ladder:

```
x1 = x
x2 = x*x
for i = k - 2 to 0:        # k bits in n, MSB (n_(k-1)) = 1
        if n_i == 0:       # bit is even
                x2 = x1*x2
                x1 = x1*x1
        else:              # bit is odd
                x1 = x1*x2
                x2 = x2*x2
return x1
```

Alternative side channel defense: blinding

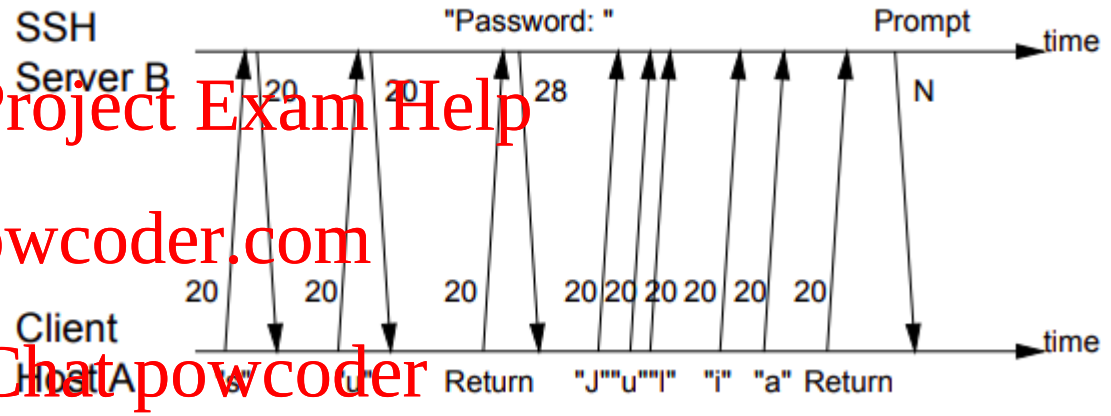
- Given c = x^e mod N
- Don’t want to compute c^d mod N (might leak d!)
- First **blind**: b = c*r^e mod N for random r (this is just (xr)^e mod N)
- Then decrypt: a = b^d mod N = (xr)^ed mod N = xr mod N
- Remove blinding: a*r^-1 mod N = xr*r^-1 mod N = x mod N
- Since attacker doesn’t know r, can’t learn d during “blinded” decryption

Other side channels?

- What other examples of side channels exist?
- How can we fix them?

Other side channels?

- EM-emission
- Sound
- Accelerometer data
- Timing of key presses
- Shared resources:
  - Cache timing
  - Bandwidth / latency
  - IPID field in IP packets



Cache side channels

- Caches improve performance by storing recently-accessed data close to the CPU
- *Potentially leaks what was recently accessed!*

1. **Attacker** fills cache:

0xA0
0xA4
0xA8
0xAC

2. **Victim process** reads from 0xC8:

0xA0
0xA4
0xC8
0xAC

0xA8 is evicted

3. **Attacker** reads:

0xA0 (52ns)
0xA4 (55ns)
0xA8 (397ns) →
0xAC (49ns)

Attacker learns **0xA8** was not in cache!
(recently evicted by another process)

AES S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

The column is determined by the least significant nibble, and the row by the most significant nibble. For example, the value $9a_{16}$ is converted into $b8_{16}$.