Hellman's TMTO Attack

Popcnt

- Before we consider Hellman's attack, consider simpler **T**ime-**M**emory **T**rade-**O**ff
- "Population count" or popcnt
  - Let x be a 32-bit integer
  - Define $\text{popcnt}(x)$ = number of 1's in binary expansion of x
- How to compute $\text{popcnt}(x)$ efficiently?

Simple Popcnt

- Most obvious thing to do is

popcnt(x) // assuming x is 32-bit value

$t = 0$

for $i = 0$ to 31

$t = t + ((x >> i) \,\&\, 1)$

next i

return t

end popcnt

- Is this the most efficient method?

More Efficient Popcnt

- Pre-compute popcnt for all 256 bytes
- Store pre-computed values in a table
- Given x, lookup its bytes in this table
  - Sum these values to find $\text{popcnt}(x)$
- Note that pre-computation is done once
- Each popcnt now requires 4 steps, not 32

More Efficient Popcnt

Initialize: table[i] = popcnt(i) for i = 0,1,…,255

```
popcnt(x) // assuming x is 32-bit word
    p = table[ x & 0xff ]
          + table[ (x >> 8) & 0xff ]
          + table[ (x >> 16) & 0xff ]
          + table[ (x >> 24) & 0xff ]
    return p
end popcnt
```

TMTO Basics

- Pre-computation
  - One-time work
  - Results stored in a table
- Pre-computation results used to make each subsequent computation faster
- Try to balance "memory" and "time"
- In general, larger pre-computation requires more initial work and larger "memory" but then each computation takes less "time"

Block Cipher Notation

- Consider a block cipher

$$C = E(P, K)$$

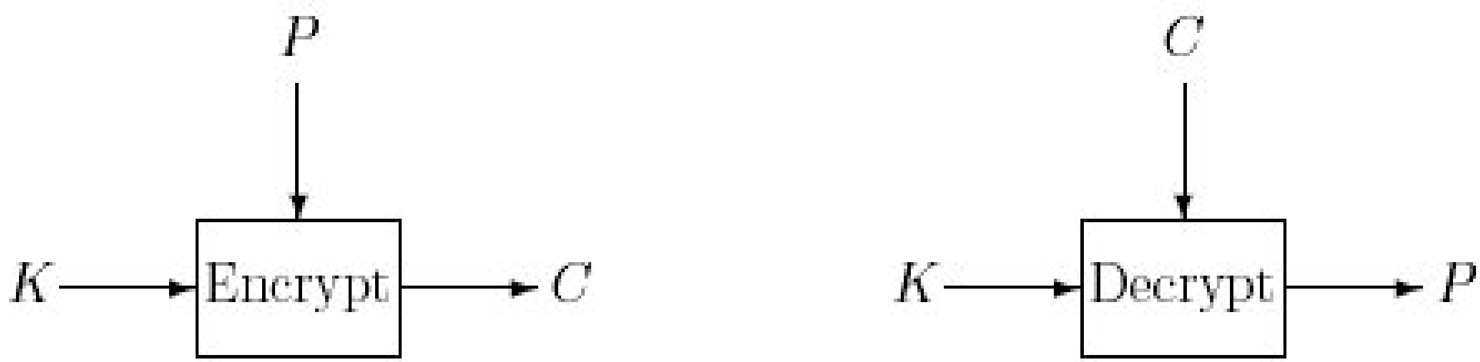
where

P is plaintext block of size n

C is ciphertext block of size n

K is key of size k

Block Cipher as Black Box



- For TMTO, treat block cipher as black box
- Details of crypto algorithm not important

Hellman's TMTO Attack

- **Chosen plaintext attack:** choose P and obtain C, where $C = E(P, K)$
- Want to find the key K
- Two "obvious" approaches
  1. **Exhaustive key search**

     "Memory" is 0, but "time" of 2^{k-1} for each attack
  2. **Pre-compute** $C = E(P, K)$ **for all keys** K

     Given C, simply look up key K in the table

     "Memory" of 2^k but "time" of 0 for each attack
- TMTO lies between 1. and 2.

Chain of Encryptions

- Assume block length n and key length k are equal: $n = k$
- Then a **chain** of encryptions is

$$SP = K_0 = \text{ Starting Point}$$

$$K_1 = E(P, SP)$$

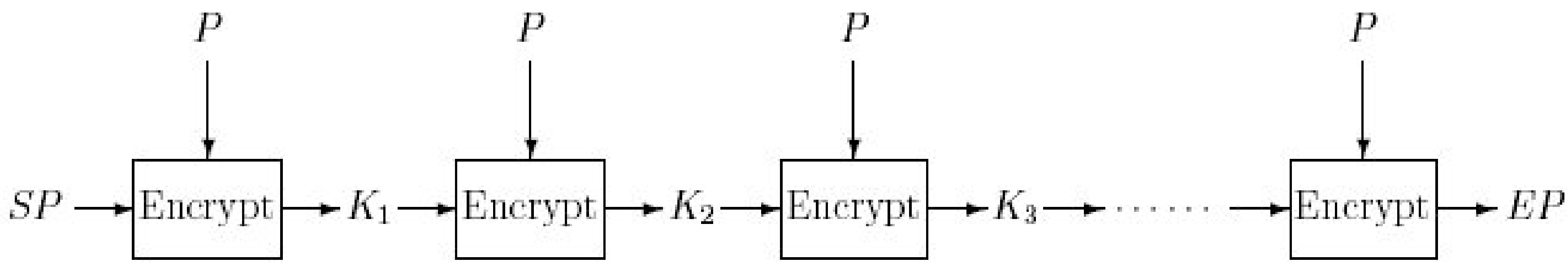
$$K_2 = E(P, K_1)$$

$$\vdots$$

$$\vdots$$

$$EP = K_t = E(P, K_{t-1}) = \text{ End Point}$$

Encryption Chain



- Ciphertext used as **key** at next iteration
- Same (chosen) **plaintext** P used at each iteration

Pre-computation

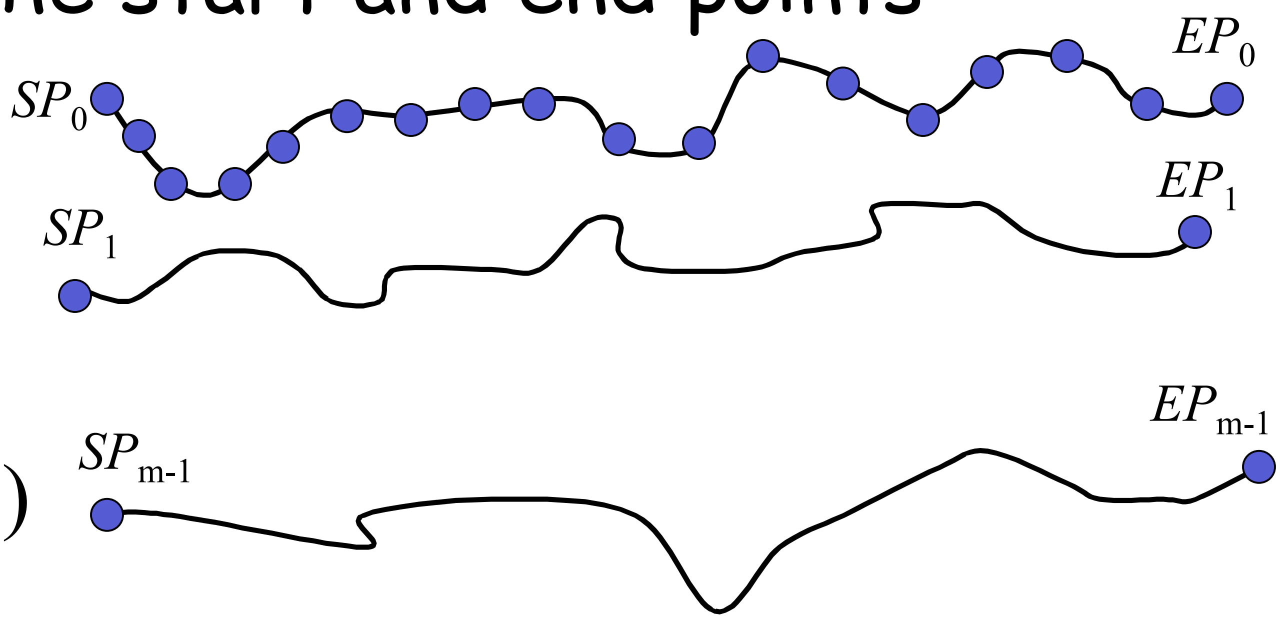
- Pre-compute m encryption chains, each of length $t+1$
- Save only the start and end points

(SP_0, EP_0)

(SP_1, EP_1)

$\vdots$

$(SP_{m\text{-}1}, EP_{m\text{-}1})$



TMTO Attack

- **Memory:** Pre-compute encryption chains and save (SP_i, EP_i) for $i = 0,1,\ldots,m-1$
  - This is one-time work
  - Must be sorted on EP_i
- To attack a particular unknown key K
  - For the same chosen P used to find chains, we know C where $C = E(P, K)$ and K is unknown key
  - **Time:** Compute the chain (maximum of t steps)

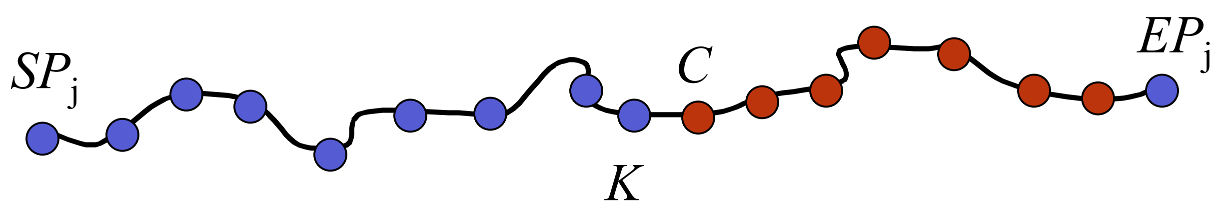
$$X_0 = C,\ X_1 = E(P, X_0),\ X_2 = E(P, X_1),\ldots$$

TMTO Attack

- Consider the computed chain

$$X_0 = C,\ X_1 = E(P, X_0),\ X_2 = E(P, X_1),\ \ldots$$

- Suppose for some i we find $X_i = EP_j$



- Since $C = E(P, K)$ key K should lie before ciphertext C in chain!

TMTO Attack

- Summary of attack phase: we compute chain

  $X_0 = C, X_1 = E(P, X_0), X_2 = E(P, X_1), \ldots$

- If for some i we find $X_i = EP_j$
- Then reconstruct chain from SP_j

  $Y_0 = SP_j, Y_1 = E(P, Y_0), Y_2 = E(P, Y_1), \ldots$

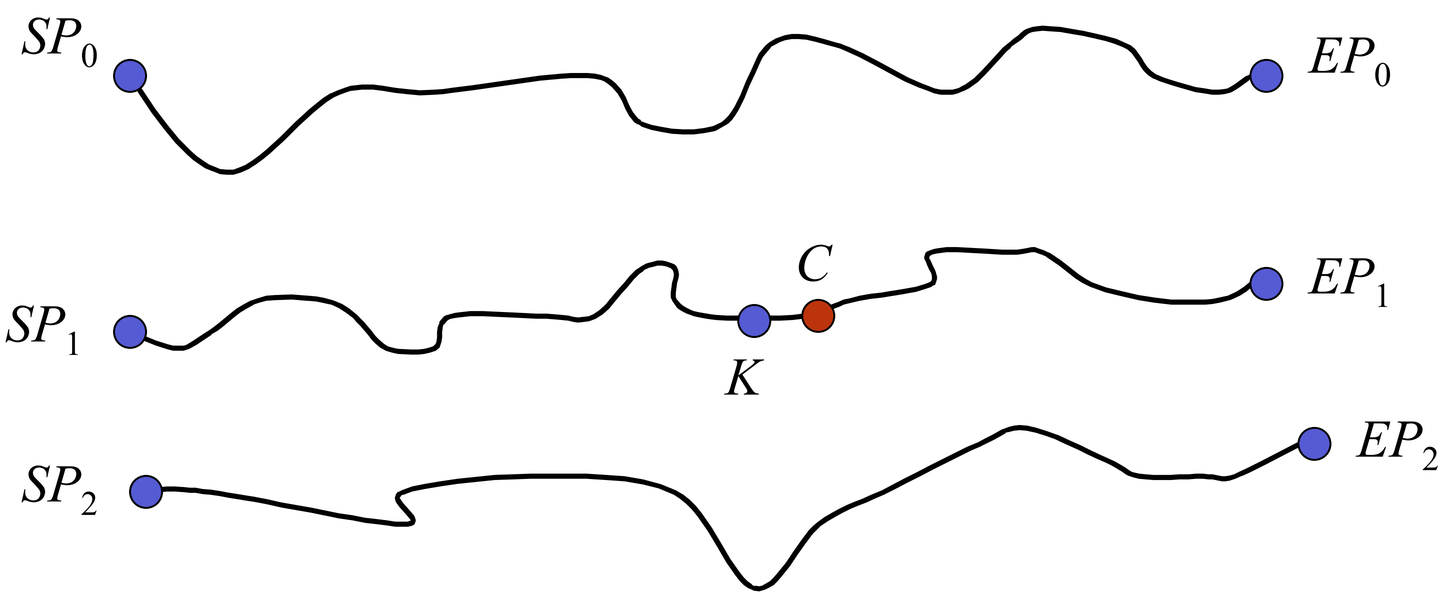
- Find $C = Y_{t-i} = E(P, Y_{t-i-1})$ (always?)
- Then $K = Y_{t-i-1}$ (always?)

Trudy's Perfect World

- Suppose block cipher has $k = 56$
  - That is, the key length is 56 bits
- Spse we find $m = 2^{28}$ chains each of length $t = 2^{28}$ and no chains overlap (unrealistic)
- **Memory:** 2^{28} pairs (SP_j, EP_i)
- **Time:** about 2^{28} (per attack)
  - Start at C, find some EP_j in about 2^{27} steps
  - Find K with about 2^{27} more steps
- Attack never fails!

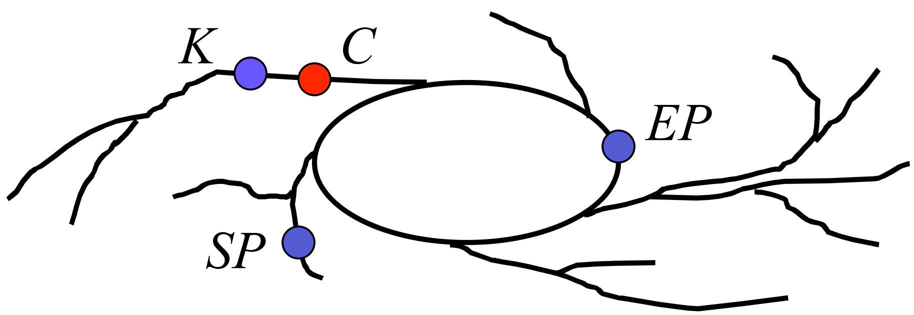
Trudy's Perfect World

- No chains overlap
- Every ciphertext C is in one chain



The Real World

- Chains are not so well-behaved!
- Chains can **cycle** and **merge**



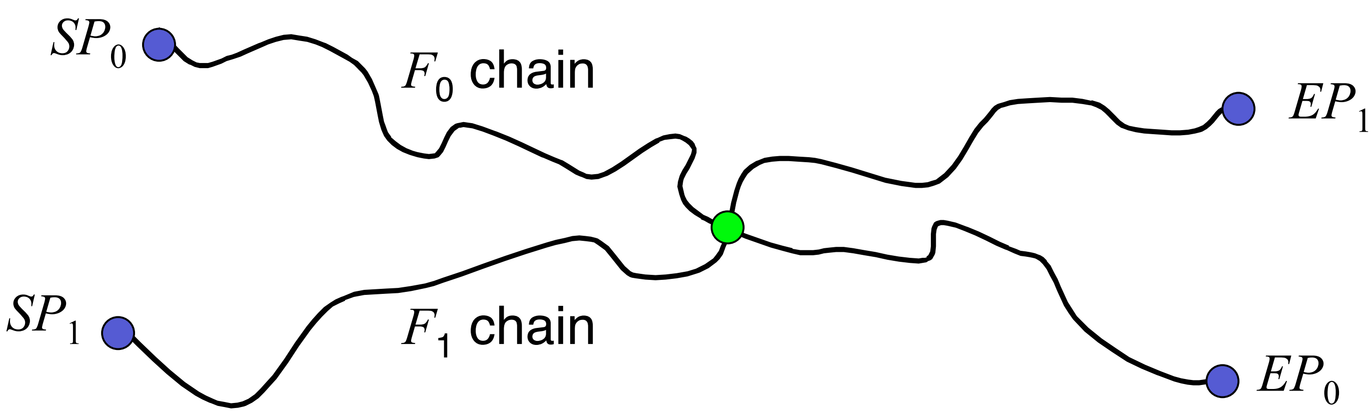
- Chain beginning at C goes to EP
- But chain from SP to EP does not give K
- Is this Trudy's nightmare?

Real-World TMTO Issues

- Merging chains, cycles, false alarms, etc.
- Pre-computation is lots of work
  - Must attack many times to amortize cost
- Success is not assured
  - Probability depends on initial work
- What if block size not equal key length?
  - This is easy to deal with
- What is the probability of success?
  - This is not so easy to compute…

To Reduce Merging

- Compute chain as $F(E(P, K_{i-1}))$ where F permutes the bits
- Chains computed using different functions can intersect, but they will **not** merge



Hellman's TMTO in Practice

- Let
  - m = random starting points for each F
  - t = encryptions in each chain
  - r = number of "tables", i.e., random functions F
- Then mtr = total pre-computed chain elements
- Pre-computation is about mtr work
- Each TMTO attack requires
  - About mr "memory" and about tr "time"
- If we choose $m = t = r = 2^{k/3}$ then probability of success is at least 0.55

Success Probability

- Throw n balls into m urns
- What is expected number of urns that have at least one ball?
- This is classic "occupancy" problem
  - See Feller, *Intro. to Probability Theory*
- Why is this relevant to TMTO attack?
  - "Urns" correspond to keys
  - "Balls" correspond to constructing chains

Success Probability

- Using occupancy problem approach
- Assuming k-bit key and m,t,r defined as previously discussed
- Then, approximately,

$$P(\text{success}) = 1 - e^{-mtr/k}$$

- An upper bound can be given that is slightly "better"

Success Probability

- Success probability

$$P(\text{success}) = 1 - e^{-mtr/k}$$

mtr	$P(\text{success})$
0	0
2^{k-5}	0.03
2^{k-4}	0.06
2^{k-3}	0.12
2^{k-2}	0.22
2^{k-1}	0.39
2^{k}	0.63
2^{k+1}	0.86
2^{k+2}	0.98
2^{k+3}	0.99
∞	1.00

Distributed TMTO

- Employ “distiguished points”
- Do not use fixed-length chains
- Instead, compute chain until some distinguished point is found
- Example of distinguished point:

$$(x_0, x_1, \ldots, x_{s-1}, \underbrace{0, 0, \ldots, 0}_{n-s})$$

Distributed TMTO

- Similar pre-computation, except we have triples:

$$(SP_i, EP_i, l_i) \text{ for } i = 0,1,\ldots,rm$$

  - Where l_i is the length of the chain
  - And r is number of tables
  - And m is number of random starting points
- Let M_i be the maximum l_j for the i^{th} table
- Each table has a fixed random function F

Distributed TMTO

- Suppose r computers are available
- Each computer deals with one table
  - That is, one random function F
- "Server" gives computer i the values F_i, M_i, C and definition of distinguished point
- Computer i computes chain beginning from C using F_i of (at most) length M_i

Distributed TMTO

- If computer i finds a distinguished point within M_i steps
  - Returns result to "server" for secondary test
  - Server searches for K on corresponding chain (same as in non-distributed TMTO)
  - False alarms possible (distinguished points)
- If no distinguished point found in M_i steps
  - Computer i gives up
  - Key cannot lie on any F_i chains
- Note that computer i does not need any SP
- Only server needs (SP_i, EP_i, l_i) for $i = 0,1,\ldots,rm$

TMTO: The Bottom Line

- Attack is feasible against DES
- Pre-computation is about 2^{56} work
- Each attack requires about
  2^{37} "memory" and 2^{37} "time"
- Attack not particular to DES
- No fancy math is required!
- Lesson: **Clever algorithms can break crypto!**