Efficient Dissection of Composite Problems, with Applications to Cryptanalysis, Knapsacks, and Combinatorial Search Problems

Itai Dinur[1], Orr Dunkelman[1,2], Nathan Keller[3] and Adi Shamir[1]

[1] Computer Science department, The Weizmann Institute, Rehovot, Israel

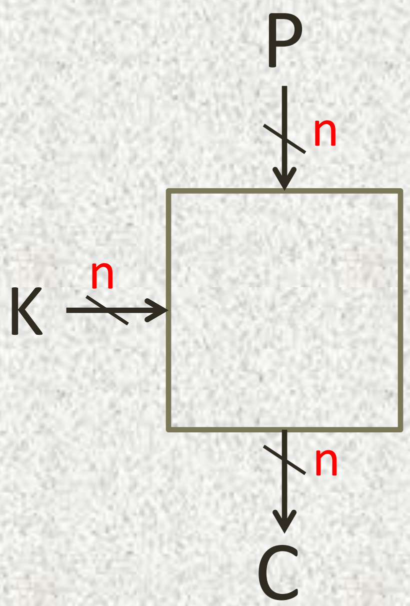
[2] Computer Science Department, University of Haifa, Israel

[3] Department of Mathematics, Bar-Ilan University, Israel

Single Encryption

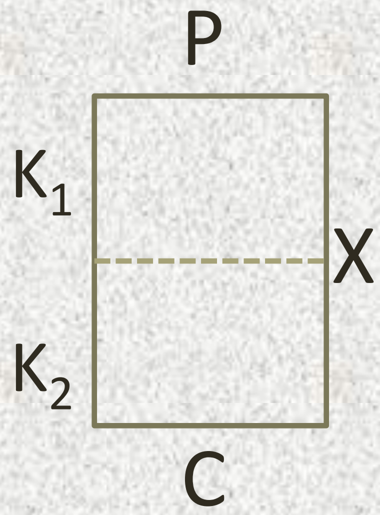
- The Basic Cryptanalytic Problem:
  - **Input**: a list of plaintext-ciphertext pairs $(P_1,C_1), (P_2,C_2),(P_3,C_3),\ldots$
  - **Goal**: find all keys K such that

$$C_1 = E_K(P_1),\ C_2 = E_K(P_2),\ldots$$



- Exhaustive Search:
  - For each n-bit value of K
    - Perform trial encryptions i.e., test whether $C_1 = E_K(P_1)$, if so test whether $C_2 = E_K(P_2)$ …
  - **Time: 2^n, Memory: constant**

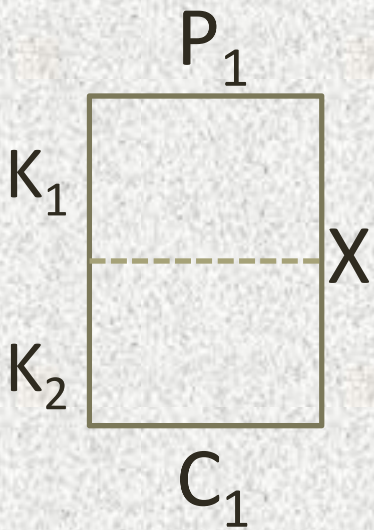
Double Encryption



- $C=E_{K_2}(E_{K_1}(P))$ with independent keys n-bit keys K_1,K_2

- Suggested following concerns about the small keys size of **DES**

MITM Attack (Hellman, Merkle '81)



K_1	X
101	000
011	010
⋮	⋮
110	111

- For each n-bit value of K_1
  - Partially encrypt P_1 and store the n-bit suggestions for X in a **sorted list**
- For each n-bit value of K_2
  - Partially decrypt C_1 and look for **matches in the list**
  - For each of the $\approx 2^n$ matches **test the full key**
- Time 2^n, memory 2^n (ignoring logarithmic factors)

Triple Encryption

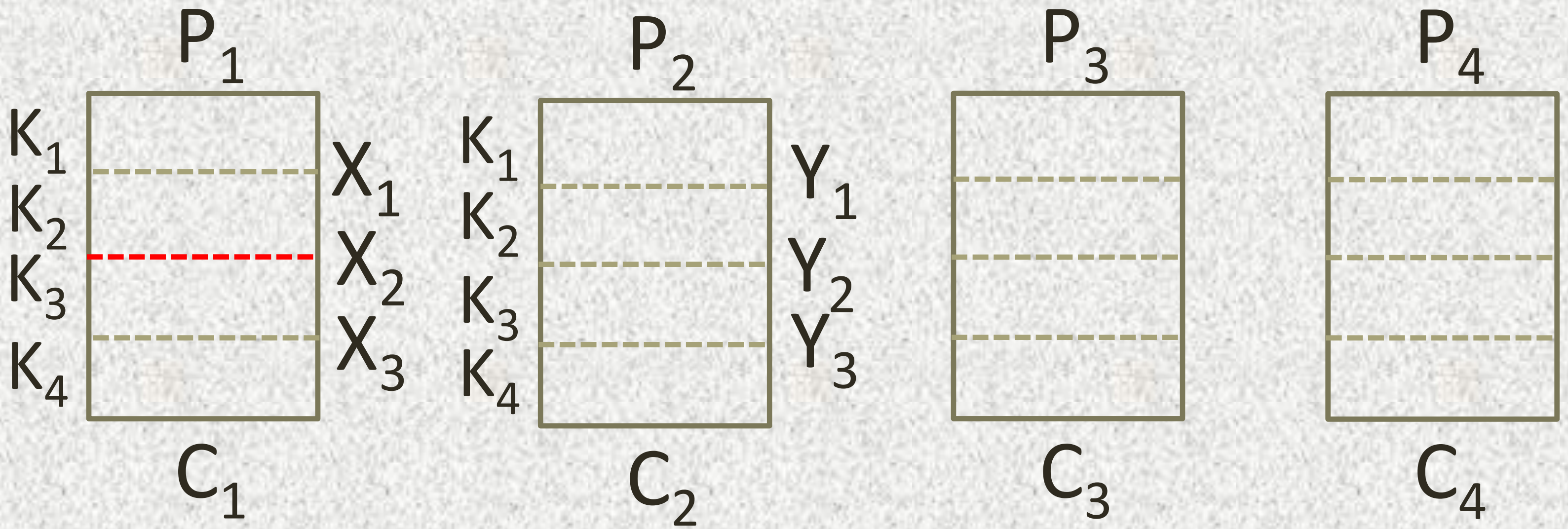
- **Triple Encryption**: $C=E_{K_3}(E_{K_2}(E_{K_1}(P)))$ with independent keys K_1,K_2,K_3
  - **Triple-DES** was used as a de-facto encryption standard from 1998 until 2001 (and even today…)

- A trivial extension of the **MITM** attack (by guessing K_3) breaks triple encryption in time 2^{2n} and memory 2^n
  - Still the best known algorithm for triple encryption

Multiple Encryption

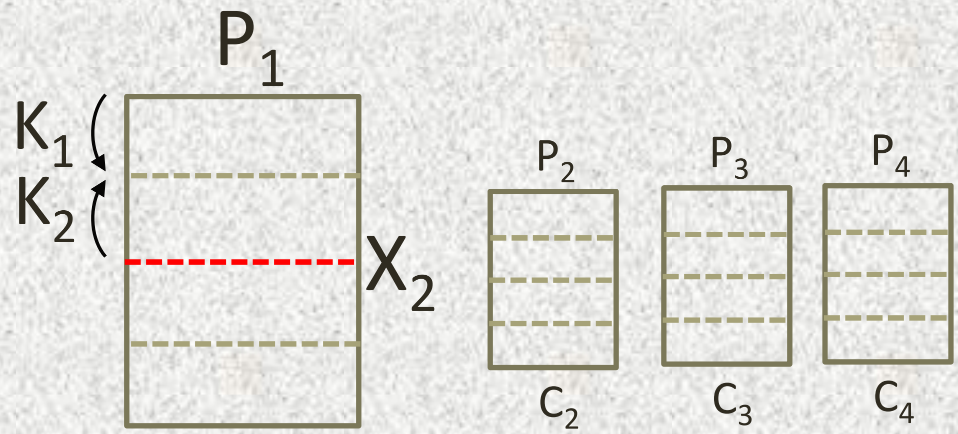
- r-fold encryption: $E_{K_r}(E_{K_{r-1}}(\ldots(E_{K_1}(P)))$ with independent keys $K_1, K_2, \ldots, K_r$
- An extension of MITM breaks r-fold encryption in time T and memory M such that $TM=2^{rn}=N$ (provided $M \le 2^{[r/2]n}$)
- Suggests an optimal time-memory tradeoff of $TM=N$

Improved Attack on 4-Fold Encryption with $M=2^n$



- For each n-bit value of X_2

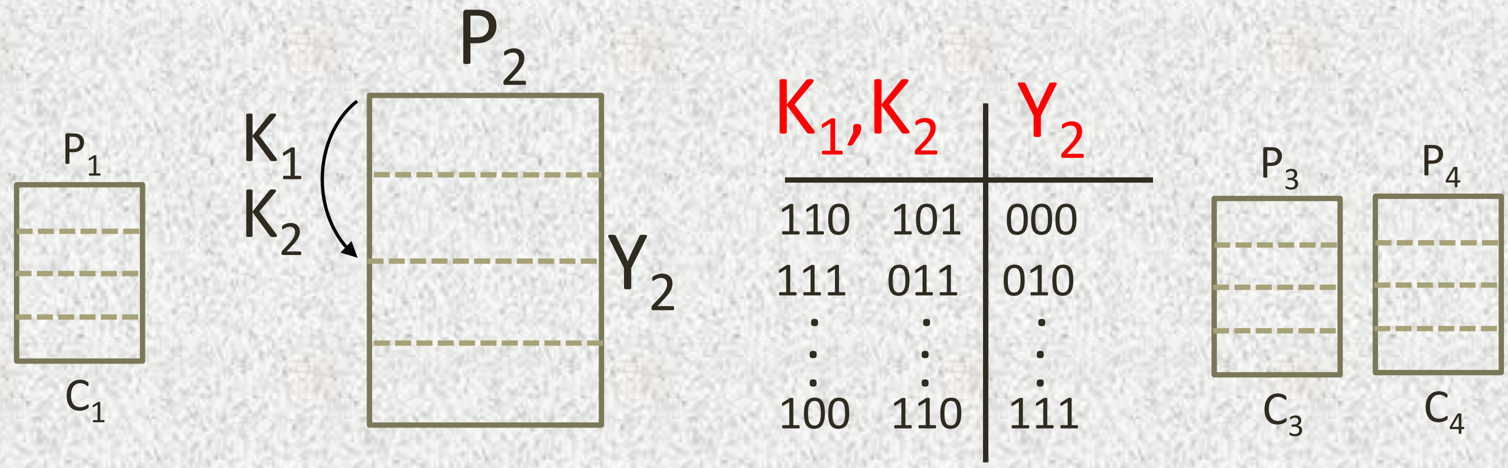
Improved Attack on 4-Fold Encryption with $M=2^n$



- For each n-bit value of X_2

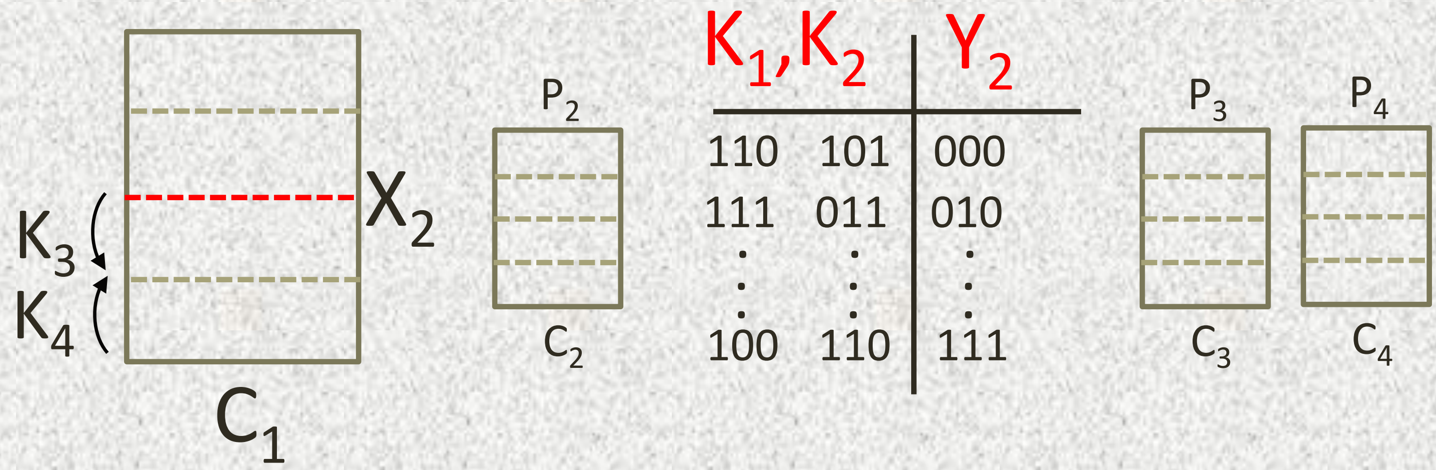
→ Given P_1,X_2 obtain $\approx 2^n$ suggestions for K_1,K_2 using a 2R MITM attack

Improved Attack on 4-Fold Encryption with $M=2^n$



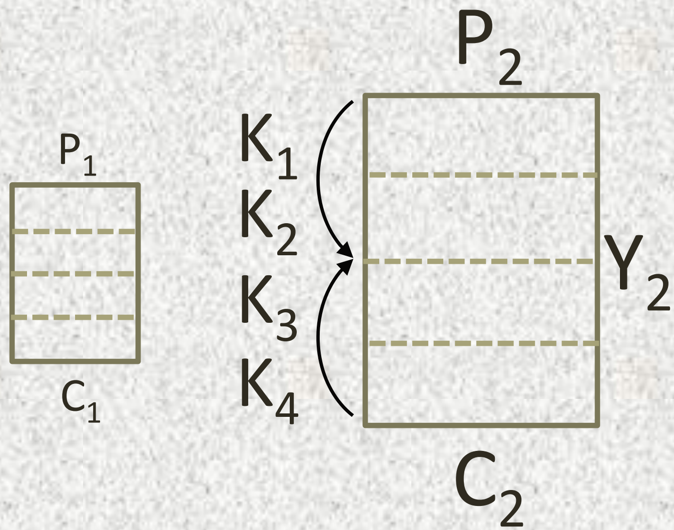
- For each n-bit value of X_2
  - Given P_1,X_2 obtain $\approx 2^n$ suggestions for K_1,K_2 using a 2R MITM attack
  - → For each suggestion, obtain Y_2 and store the triplet in a sorted list

Improved Attack on 4-Fold Encryption with $M=2^n$

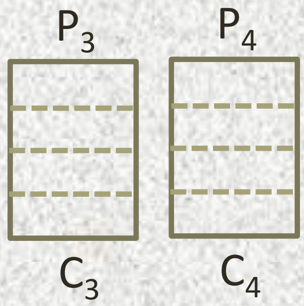


- For each n-bit value of X_2
  - Given P_1,X_2 obtain $\approx 2^n$ suggestions for K_1,K_2 using a 2R MITM attack
  - For each suggestion, obtain Y_2 and store the triplet in a sorted list
  - → Given X_2,C_1 obtain $\approx 2^n$ suggestions for K_3,K_4 using a 2R MITM attack

Improved Attack on 4-Fold Encryption with $M=2^n$

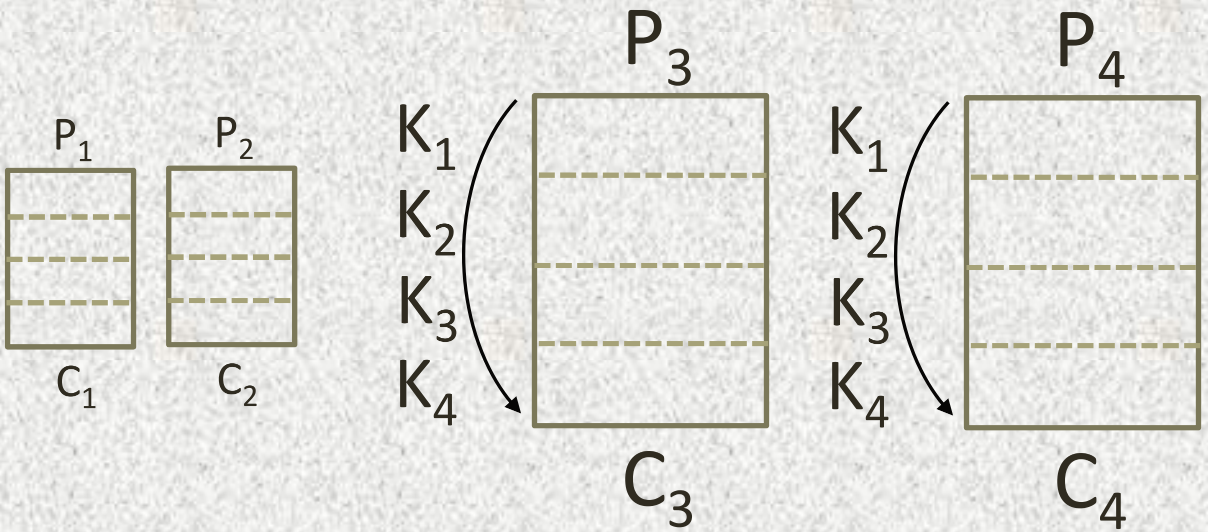


K_1,K_2		Y_2
110	101	000
111	011	010
⋮	⋮	⋮
100	110	111



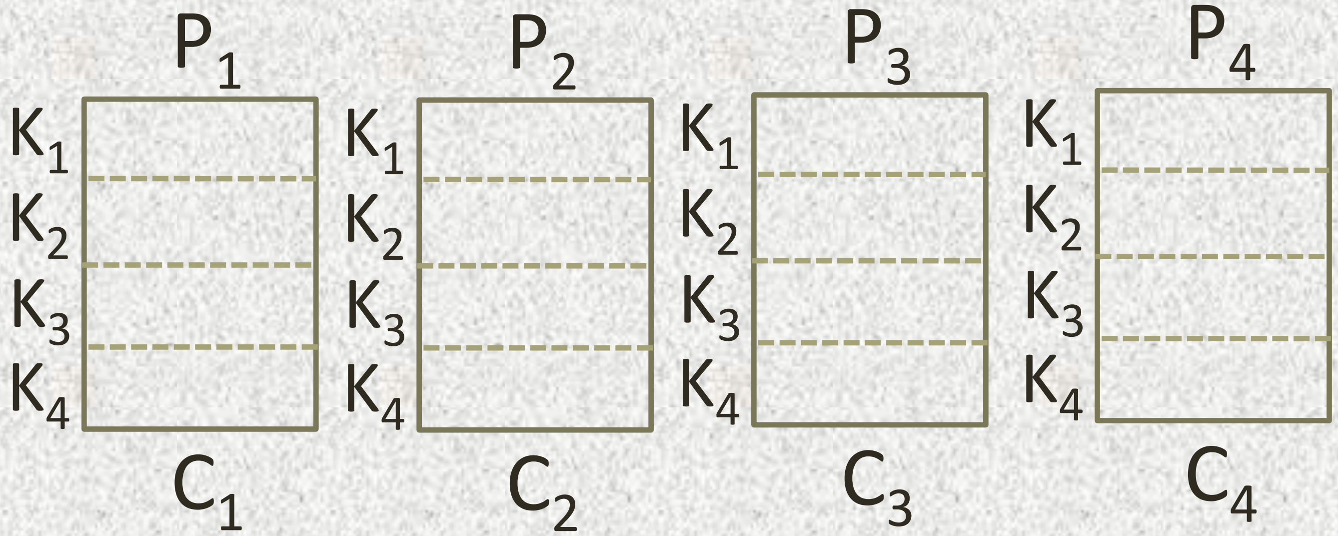
- For each n-bit value of X_2
  - Given P_1,X_2 obtain $\approx 2^n$ suggestions for K_1,K_2 using a 2R MITM attack
  - For each suggestion, obtain Y_2 and store the triplet in a sorted list
  - Given X_2,C_1 obtain $\approx 2^n$ suggestions for K_3,K_4 using a 2R MITM attack
  - → For each suggestion, obtain Y_2 and match with the stored list

Improved Attack on 4-Fold Encryption with $M=2^n$



- For each n-bit value of X_2
  - Given P_1,X_2 obtain $\approx 2^n$ suggestions for K_1,K_2 using a 2R MITM attack
  - For each suggestion, obtain Y_2 and store the triplet in a sorted list
  - Given X_2,C_1 obtain $\approx 2^n$ suggestions for K_3,K_4 using a 2R MITM attack
  - For each suggestion, obtain Y_2 and match with the stored list
  - → For each of the $\approx 2^n$ matches **test the full key** using (P_3,C_3) and (P_4,C_4)

Improved Attack on 4-Fold Encryption with $M=2^n$



- For each n-bit value of X_2
  - Given P_1,X_2 obtain $\approx 2^n$ suggestions for K_1,K_2 using a 2R MITM attack
  - For each suggestion, obtain Y_2 and store the triplet in a sorted list
  - Given X_2,C_1 obtain $\approx 2^n$ suggestions for K_3,K_4 using a 2R MITM attack
  - For each suggestion, obtain Y_2 and match with the stored list
  - For each of the $\approx 2^n$ matches **test the full key** using (P_3,C_3) and (P_4,C_4)
- Time 2^{2n}, memory 2^n (the same as triple-encryption!)

Increasing r Further

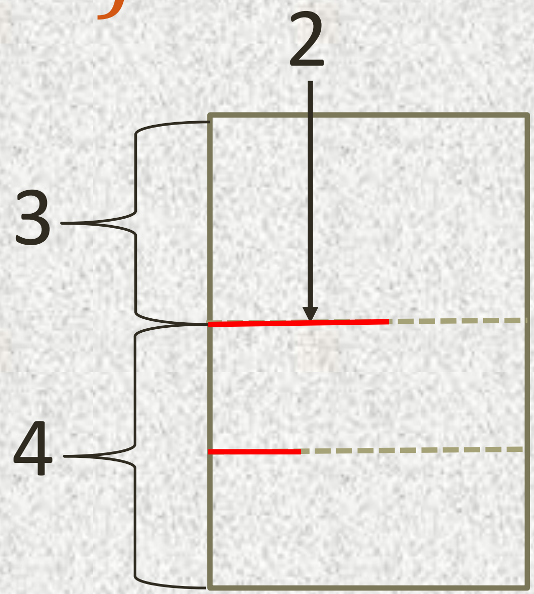
- We obtained $TM=2^{3n}$ (instead of 2^{4n}) for $r=4$

- What happens when we increase r further?
- We first fix $M=2^n$ and try to minimize T

r	1	2	3	4	5	6	7	8	…
T	2^n	2^n	2^{2n}	~~2^{3n}~~	~~2^{4n}~~	~~2^{5n}~~	~~2^{6n}~~	~~2^{7n}~~	
				2^{2n}	2^{3n}	2^{4n}	~~2^{5n}~~	~~2^{6n}~~	

Surprisingly Efficient Attack on 7-Fold Encryption (a 7r attack)

- Split the 7r cipher into two subciphers, a 3r top part and a 4r bottom part
- Guess 2 intermediate encryption values in the middle (one for (P_1,C_1) and one for (P_2,C_2))
  - Apply a 3r attack to the top part and **store** the 2^n returned suggestions
  - Apply the 4r attack to the bottom part and test the returned keys **on the fly**



Analysis of the Attack

- We guess 2n bits in the middle
  - The top 3r attack takes 2^{2n} time and 2^n memory
  - The bottom 4r attack takes 2^{2n} time and 2^n memory
- The total complexity is $T=2^{4n}$ (instead of 2^{6n})
- We obtain $TM=2^{5n}$ (instead of 2^{7n})

Extending the 7r Attack

- Our 7r attack divides the cipher **asymmetrically** into a top and bottom part

r	1	2	3	4	5	6	7	8	…
T	2^n	2^n	2^{2n}	~~2^{3n}~~	~~2^{4n}~~	~~2^{5n}~~	~~2^{6n}~~	~~2^{7n}~~	
				2^{2n}	2^{3n}	2^{4n}	~~2^{5n}~~	~~2^{6n}~~	
							2^{4n}	2^{5n}	

- Can be extended recursively by dividing the cipher **asymmetrically** into subciphers

Constructing Asymmetric Algorithms

- Using the asymmetric recursion, we construct a "magic sequence" of the "turning points"

  Magic={4,7,11,16,22,29,37,46,…}
- The algorithm becomes **increasingly more efficient** compared to the standard MITM
  - For $r=4$, we have $T=2^{2n}$ (compared to $T=2^{3n}$)
  - For $r=7$, we have $T=2^{4n}$ (compared to $T=2^{6n}$)
  - For $r=11$, we have $T=2^{7n}$ (compared to $T=2^{10n}$)…
- We obtain an asymptotic time complexity of

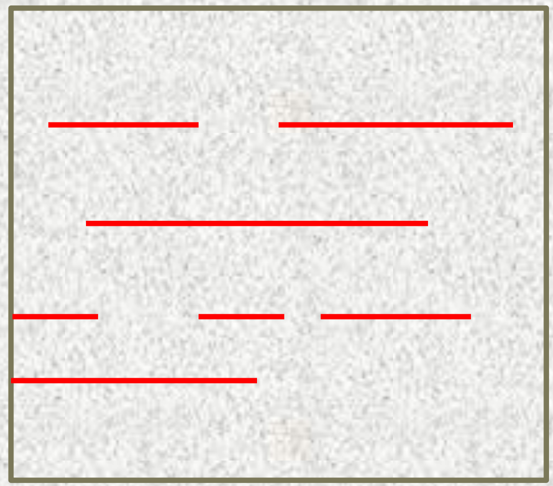
  $T \approx 2^{n(r-\sqrt{2r})}$
- The algorithms generalize to any amount of memory

Where does the asymmetry come from?

- Most recursive algorithms divide the problem symmetrically to avoid bottlenecks
- However, there is asymmetry between the top and bottom subciphers
  - In the top part, we store all remaining suggestions **in memory** -> at most 2^n suggestions can remain
  - In the bottom part, we can check the key suggestions **on the fly** -> no restriction on their number!
- Hence, it is better to have more rounds in the bottom part!

Dissection Algorithms

- We obtain a new class of algorithms which we call **dissection** algorithms
- We perform "cuts" of different sizes in carefully chosen places of the encryption structure



Composite Problems

- A **composite problem**
  - We are given the initial value(s) and the final value(s) of a cascade of r steps
  - In each step, one of a list of possible transformations was applied
  - The goal: Find out, which transformation was applied in each step (i.e., find all possible options)
- Clearly, r-fold encryption is a composite problem

Application to Knapsacks

- **Modular Knapsack Problem:**
  - **Input**: A list of n integers $\{a_1,a_2,\ldots,a_n\}$ of n bits each, and a target integer S
  - **Goal**: Find a vector $\varepsilon=\{\varepsilon_1,\varepsilon_2\ldots\varepsilon_n\}$ where $\varepsilon_i\in\{0,1\}$ such that $S=\sum_{1\le i\le n}(\varepsilon_i\cdot a_i) \bmod 2^n$

- How do we apply the dissection techniques to the Knapsack problem?

Representing Knapsack as a Block Cipher

P

$+(\varepsilon_1 \cdot a_1)$

$+(\varepsilon_2 \cdot a_2)$

$\vdots$

$+(\varepsilon_n \cdot a_n)$

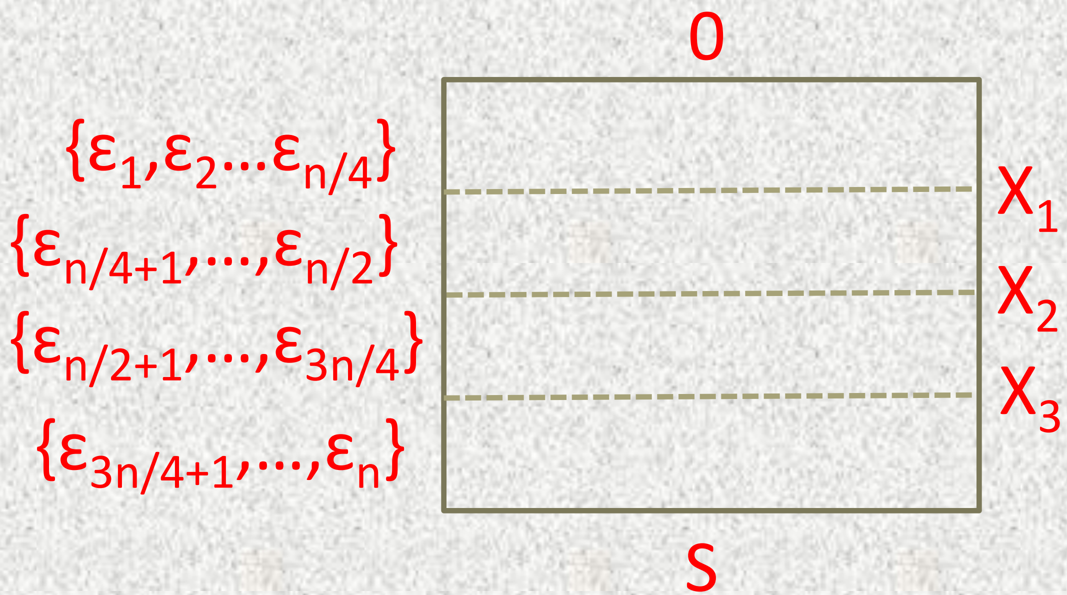
$\varepsilon = \{\varepsilon_1, \varepsilon_2 \ldots \varepsilon_n\}$

$$C = P + \sum_{1 \le i \le n} (\varepsilon_i \cdot a_i) \pmod{2^n}$$

- We fix the plaintext to be the 0 n-bit vector, the ciphertext to be S
- The knapsack problem **reduces** to recovering the key of this block cipher, given **one** plaintext-ciphertext pair

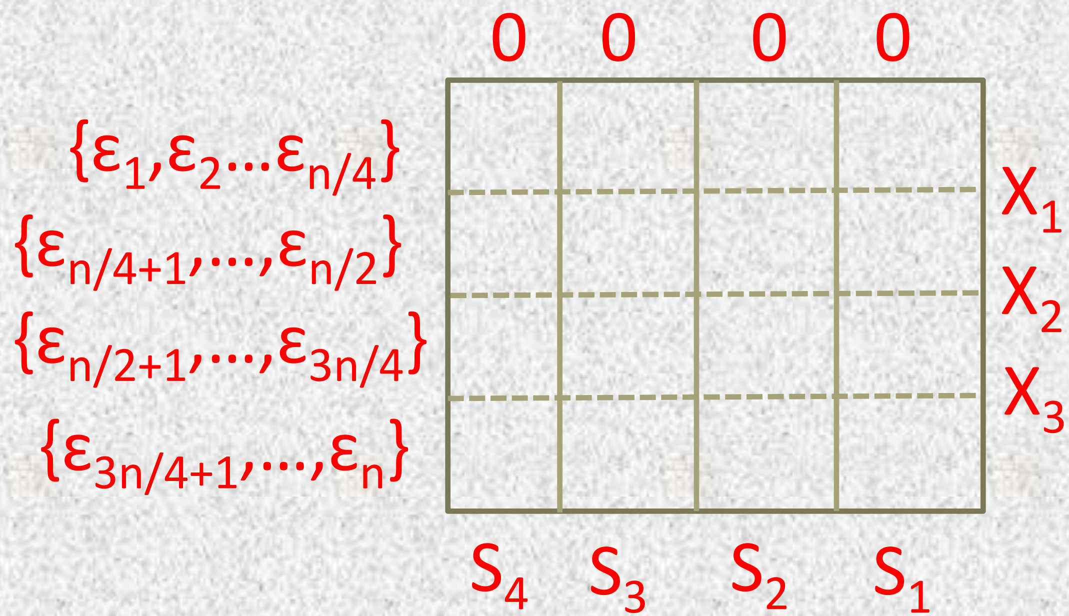
Representing Knapsack as 4-Fold Encryption

- We split the knapsack to 4 independent knapsacks by splitting the generators and defining $S=\sigma_1+\sigma_2+\sigma_3+\sigma_4 \pmod{2^n}$
- $X_i=\sum_{1\le j\le i}(\sigma^j)$



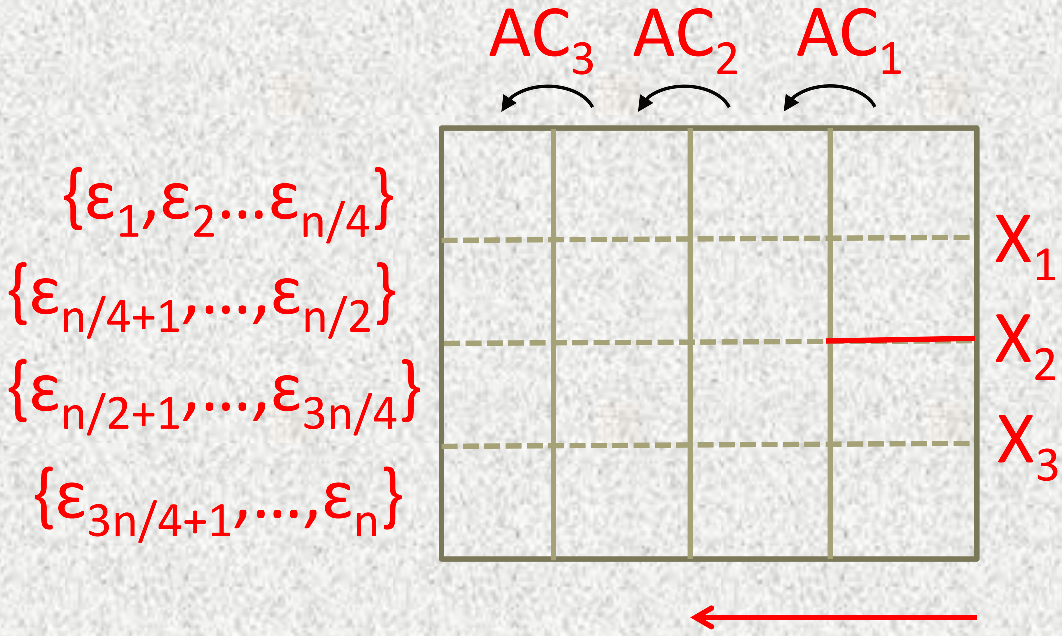
Representing Knapsack as 4-Fold Encryption

- **Problem:** In r-fold encryption, we have r “small” plaintexts -> can efficiently guess intermediate values. Here we have a single “big” plaintext
- **Solution:** Split the “block cipher” also **vertically** into n/4-bit blocks



Representing Knapsack as 4-Fold Encryption

- **Problem:** Dependency between the "vertical" chunks through addition carries

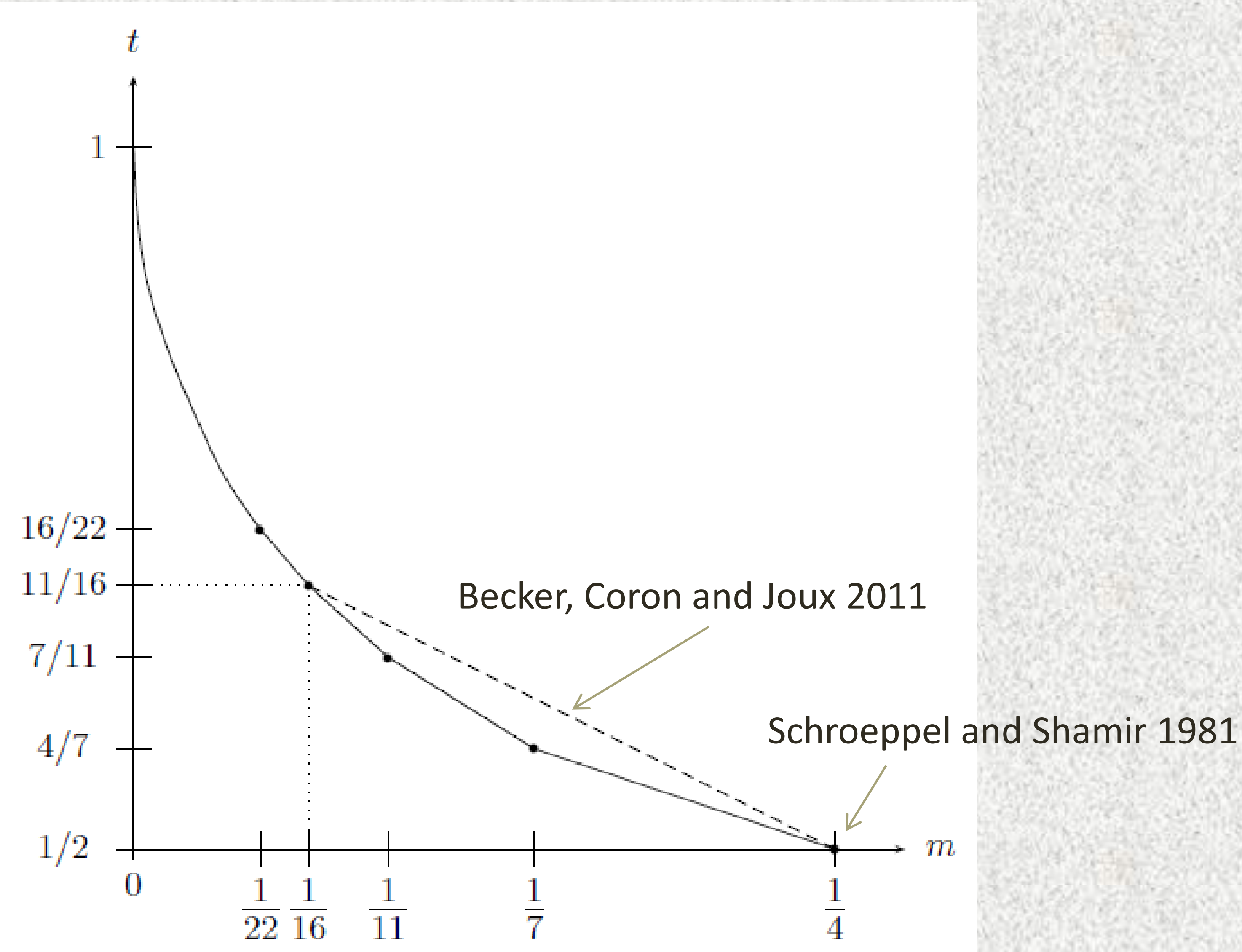


- **Solution:** Guess the intermediate encryption values in their **natural order** (from right to left)

Representing Knapsack as 4-Fold Encryption

- **Conclusion:** We can apply to knapsacks the algorithm for r-fold encryption, for any r
- We choose r according to the amount of **available memory**, in order to optimize the running time of the dissection algorithms

Time-Memory Tradeoff for Knapsacks



Examples of Other Composite Problems

- **Rubik's cube** – find a shortest solution given an initial state
- **The matching phase in rebound attacks** on hash functions
- **Card Shuffling**
- etc…

Probabilistic Algorithms for MITM

- Until now we only considered algorithms that are **guaranteed** to return all solutions
- In the second half of the paper, we combine our **dissection** algorithms with the probabalistic **Parallel Collision Search** (Van Oorschot and Wiener, CRYPTO 1996)
- We obtain significantly improved attacks for very small amounts of memory

Conclusions

- We improved the best known algorithms for multiple encryption
- Our techniques allow us to improve the **best known** algorithms for the knapsack problem **with small memory**
- These techniques are applicable to other **composite problems** that have nothing to do with cryptography

Open Problems

- Are our results optimal?
  - Can you improve our 7r attack?
- Prove **lower bounds** for composite problems
  - In particular, prove that $T \geq N^{1/2}$
- Our algorithms use the **smallest number** of P/C pairs. Can you improve the attacks by using slightly more data?
- Find additional applications to dissection algorithms

Thanks for listening!