Mathematical Cryptanalysis of RSA

Breaking RSA

- In general cases, only possible by factorization of very large public modulus N – requires subexponential time, at best. After factorization, we can obtain private key d and decrypt any message on the channel.
- However, in real situations, factorization of public modulus N and obtaining the plaintext is not practically possible unless it is a special weak case. Thus, all **mathematical attacks** on RSA to obtain either the plaintext or the private key are possible on **special weak cases**.
- Many such weak implementations of RSA are found in practical use.

Attacks we performed

- Hastad Broadcasting Attack
- Franklin-Reiter Related Message Attack
- Weiner’s Low Private Exponent Attack
- Pollard’s p-1 Factorization Attack

Low Public Exponent (e) Attacks

- Hastad Broadcasting Attack
- Franklin-Reiter Related Message Attack

Hastad Broadcasting Attack

Hastad Broadcasting Attack

Principle

Let Alice send out the same message M to multiple parties with same public exponent and different moduli (e, N_1), (e, N_2), …., (e, N_n). The encrypted message for each of the parties are $c_1 = M^e \bmod N_1$, $c_2 = M^e \bmod N_2$, … , $c_n = M^e \bmod N_n$.

Using Chinese Remainder Theorem, we can find:

$X = M^e \bmod N_1 . N_2 . N_3 . \ldots . N_n$

If $M^e < N_1 . N_2 . N_3 . \ldots . N_n$, we can easily find the original message by finding $X^{1/e}$.

Hastad Broadcasting Attack

Chinese Remainder Theorem

$c_1 \equiv M^e \bmod N_1$, $c_2 \equiv M^e \bmod N_2$, $c_3 \equiv M^e \bmod N_3$

Given e, c_1, c_2, c_3, N_1, N_2, N_3, we can compute:

$N = N_1 \times N_2 \times N_3$

$m_i = N / N_i$ for i = 1,2,3

$y_i = m_i^{-1} \bmod N_i$ for i = 1,2,3

$X = M^e \bmod N = (c_1.m_1.y_1 + c_2.m_2.y_2 + c_3.m_3.y_3) \bmod N$

Hastad Broadcasting Attack

Ways to avoid

- Use large public exponent ($e > 2^{16} = 65{,}536$). This will require Eve to need very large number of congruences to solve the CRT problem.
- Do not send the exact same message to many recipients – make each message unique (for instance, by including a timestamp or adding a *random* padding) [does not always ensure security]

Hastad proved that adding a *linear* padding to the plaintext before encryption and sending to each party does not protect against this attack! Hence, padding *must be random.*

Franklin-Reiter Related Message Attack

Franklin-Reiter Attack

Principle

Let Alice send out 2 different messages M_1, M_2 to Bob with public key (e , N) such that M_1 and M_2 are related to each other by some ***known*** linear polynomial equation $f(x) = a*x + b$ (b ≠ 0) i.e. $M_1 = f(M_2)$.

Given e, N, c_1, c_2, f :

$c_1 \equiv M_1^e \bmod N \Rightarrow c_1 \equiv f(M_2)^e \bmod N \Rightarrow M_2$ is a root of polynomial $g_1(x) = f(x)^e - c_1$ in Z_N

$c_2 \equiv M_2^e \bmod N \Rightarrow M_2$ is a root of polynomial $g_2(x) = x^e - c_2$ in Z_N

$\therefore$ $(x - M_2)$ is a factor of both g_1 and g_2.

M_2 can be obtained from the result of $gcd(g_1, g_2)$ and M_1 can be obtained by computing $f(M_2)$.

Franklin-Reiter Attack

Applicable when

- If two messages differ only from a known fixed value Δ and are RSA encrypted using the same modulus N.
    - ❖ texts differing only from their date of compilation
    - ❖ retransmission of a message with a new ID number due to an error

Ways to avoid

- Use large public exponent e ($e > 2^{32}$)

Weiner’s Low Private Exponent Attack

Weiner's Attack

Principle

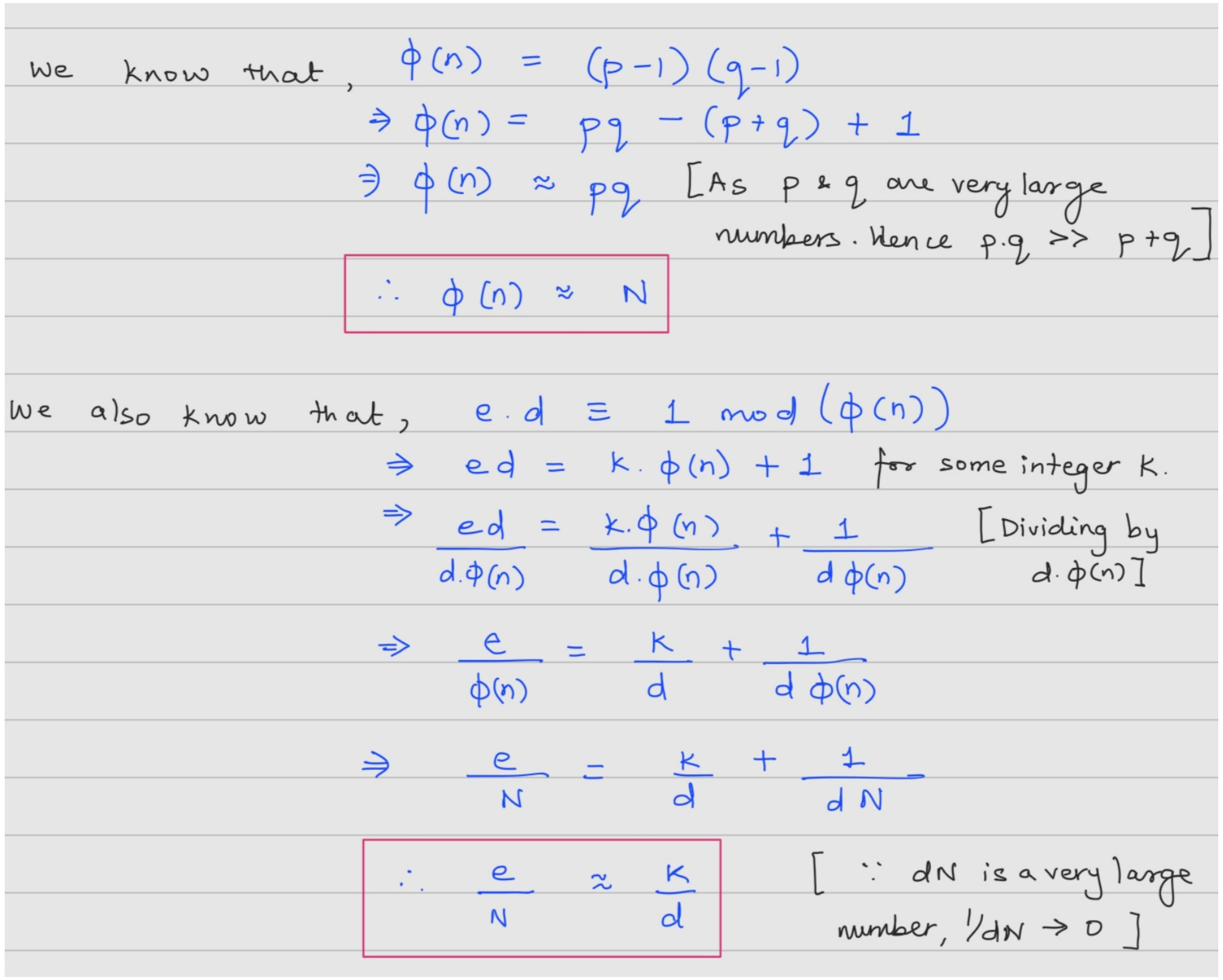
Let there be an entity with public key (e, N) and low private key d (**$d<1/3*N^{1/4}$**).

It is shown that $e / N \approx k / d$ for some integer k.

Hence, if we can approximate e / N using some rational number k / d, there is a chance that the denominator d is our decryption exponent.

We try to obtain rational approximations of e / N by method of continued fractions and iteratively checking if the denominator d is correct.

Proof



We know that, $\phi(n) = (p-1)(q-1)$

$\Rightarrow \phi(n) = pq - (p+q) + 1$

$\Rightarrow \phi(n) \approx pq$ [As p & q are very large numbers. Hence $p.q >> p+q$]

$$\boxed{\therefore \phi(n) \approx N}$$

We also know that, $e.d \equiv 1 \bmod (\phi(n))$

$\Rightarrow ed = k.\phi(n) + 1$ for some integer K.

$\Rightarrow \frac{ed}{d.\phi(n)} = \frac{k.\phi(n)}{d.\phi(n)} + \frac{1}{d\,\phi(n)}$ [Dividing by $d.\phi(n)$]

$\Rightarrow \frac{e}{\phi(n)} = \frac{K}{d} + \frac{1}{d\,\phi(n)}$

$\Rightarrow \frac{e}{N} = \frac{K}{d} + \frac{1}{dN}$

$$\boxed{\therefore \frac{e}{N} \approx \frac{K}{d}}$$

[$\because$ dN is a very large number, $1/dN \to 0$]

Weiner’s Attack

Ways to avoid

- Use large private exponent d (*$d > 1/3 * N^{1/4}$*).
- Even if there is need for faster decryption or limited computational power (such as in smart cards) to perform modular exponentiation, CRT can be used for faster decryption with sufficiently large value of d.
- Use a very large value of encryption key, *$e' = e + c.phi(N)$* for some large *c*. Then, *k* in the rational approximation *k/d* is so large that Weiner’s attack is not practical anymore.

Pollard’s p-1 Factorization Attack

Pollard’s p-1 Attack

Principle

Let p, q be two large prime numbers that form the public modulus $N = pq$.

By Fermat’s Little Theorem:

$a^{k(p-1)} \equiv 1 \bmod p$ when a and p are co-prime.

$\Rightarrow$ $a^{k(p-1)} - 1 = p.r$ for some integer r

$\Rightarrow$ $gcd(a^{k(p-1)} - 1 , N) = p$

Useful only when $p-1$ can be factorized into primes smaller than a chosen fixed prime i.e. it is power-smooth for a chosen prime number.

Pollard's p-1 Attack

Ways to avoid

- Use strong primes p and q such that $p\text{-}1$ and $q\text{-}1$ have large a prime factor. However, It takes more time to generate (provably) strong primes.

4-Round SPN Implementation and Linear Cryptanalysis

Substitution-Permutation Network

An SPN is a series of linked mathematical operations used in block cipher algorithms, such as AES.

Working of a 4-round SPN:

Such a network takes a block of plaintext (16 bits) and the key as inputs and applies several layers of substitution boxes (S-boxes) and permutation boxes (P-boxes) to produce the ciphertext block.

S-Box:

It performs one-to-one substitution of the block of input bits by another block of bits, giving us the output of the S-box.

Hexadecimal representation of the S-box used in our code:

input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
output	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

P-Box:

It takes in the output of the S-box of the corresponding round and performs bit-shuffling to permute/transpose bits

Permutation used in our code:

input	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
output	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16

Key Scheduling Algorithm:

- For a 32-bit master key and a 4-round SPN, we need to produce 5 16-bit subkeys.
- For $1 \leq r \leq 5$, define K^r to consist of 16 consecutive bits of K, beginning with K_{4r-3}

$$K = 0011\ 1010\ 1001\ 0100\ 1101\ 0110\ 0011\ 1111$$

Then the round keys are as follows:

$$K^1 = 0011\ 1010\ 1001\ 0100$$

$$K^2 = 1010\ 1001\ 0100\ 1101$$

$$K^3 = 1001\ 0100\ 1101\ 0110$$

$$K^4 = 0100\ 1101\ 0110\ 0011$$

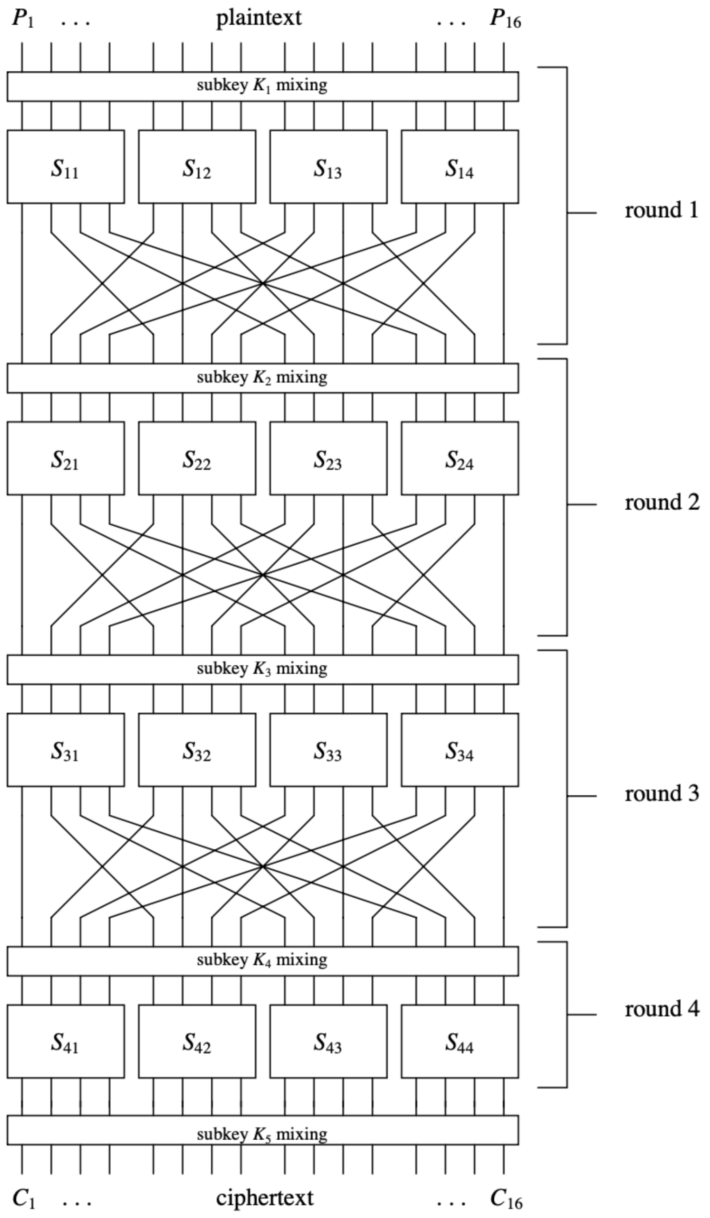
$$K^5 = 1101\ 0110\ 0011\ 1111$$

Algorithm of our implementation of the 4-round an SPN:

$\left.\begin{array}{l}\text{•The block of } \textit{plaintext} \text{ P gets XOR-ed with subkey } K^n \\ \text{•The corresponding output goes through the S-box and the} \\ \text{bits gets substituted} \\ \text{•The output of the S-box goes into the P-box as the input.}\end{array}\right\}$ (n-1) rounds

•nth round:

- ❖The output of the (n-1)th P-box gets XOR-ed with the subkey K^4
- ❖It then goes through the S-box
- ❖The output of the S-box doesn't go through a P-box, but gets XOR-ed with the last subkey K^5 which then gives us the *ciphertext*



P_1 . . . plaintext . . . P_{16}
subkey K_1 mixing
S_{11} S_{12} S_{13} S_{14}
round 1
subkey K_2 mixing
S_{21} S_{22} S_{23} S_{24}
round 2
subkey K_3 mixing
S_{31} S_{32} S_{33} S_{34}
round 3
subkey K_4 mixing
S_{41} S_{42} S_{43} S_{44}
round 4
subkey K_5 mixing
C_1 . . . ciphertext . . . C_{16}

Linear Cryptanalysis

Linear cryptanalysis is a known plaintext attack, in which the attacker studies probabilistic linear relations (linear approximations) among parity bits of the plaintext, the ciphertext and the hidden key.

Approach:

- We determine expressions which have a high or low probability of occurrence. That is, the probability of those expressions to be true should have a high deviation from ½.
- The expressions are of the form:

$X_{i_1} \oplus X_{i_2} \oplus \ldots \oplus X_{i_u} \oplus Y_{j_1} \oplus Y_{j_2} \oplus \ldots \oplus Y_{j_v} = 0$,where X_i is the ith bit of input X of the S-box and Y_j is the jth bit of output Y of the S-box

- Linear cryptanalysis exploits the most biased linear relation between its input and output. If the bits for the expression were randomly chosen and put in the expression, the probability of the expression to be true would be ½. But the cipher's poor randomization abilities are exploited, the further away that a linear expression is from holding with a probability of 1/2, the better the cryptanalyst is able to apply linear cryptanalysis.

- We calculate the *linear probability bias*, which is the deviation of the probability of a linear expression from ½, that is, if P_L is the probability of an expression for a randomly chosen plaintext and its corresponding ciphertext, the absolute probability bias = $| P_L - ½ |$

Piling-Up Principle:

For n independent, random binary variables, $X_1, X_2, \ldots X_n$,

$$\Pr(X_1 \oplus \ldots \oplus X_n = 0) = 1/2 + 2^{n-1} \prod_{i=1}^{n} \varepsilon_i$$

or, equivalently,

$$\varepsilon_{1,2,\ldots,n} = 2^{n-1} \prod_{i=1}^{n} \varepsilon_i$$

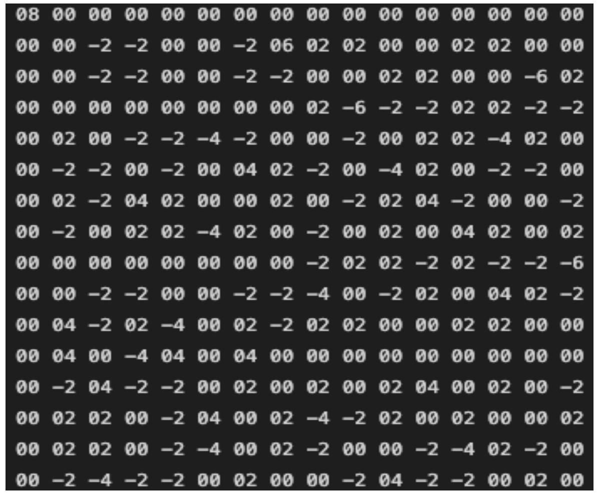
where $\varepsilon_{1,2,\ldots,n}$ represents the bias of $X_1 \oplus \ldots \oplus X_n = 0$.

Creation of the Linear Approximation Table (LAT) of the S-Box:

- We consider the sum of specific bits of the input and output that have a *strong bias*.
- These bits are selected using masks *a* and *b* for the input X and the output Y of the S-Box respectively. The masks have a 1 for the bits that we consider and 0 otherwise.
- For any pair of masks *(a,b)*, the LAT contains the bias of the equation:

$$a . x \oplus b . S(x) = 0$$ where a,b are the masks, x is the input of the S-box and S(x) is the corresponding output of the S-box.

- Each element in LAT is corresponding to values of a and b (binary representations: $a_1a_2a_3a_4$ and $b_1b_2b_3b_4$)
- Each element in the LAT represents the number of matches between the linear equation represented in hexadecimal as "Input Sum" and the sum of the output bits represented in hexadecimal as "Output Sum" minus 8.
- LAT formed by the code:



```
08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 -2 -2 00 00 -2 06 02 02 00 00 02 02 00 00
00 00 -2 -2 00 00 -2 -2 00 00 02 02 00 00 -6 02
00 00 00 00 00 00 00 00 02 -6 -2 -2 02 02 -2 -2
00 02 00 -2 -2 -4 -2 00 00 -2 00 02 02 -4 02 00
00 -2 -2 00 -2 00 04 02 -2 00 -4 02 00 -2 -2 00
00 02 -2 04 02 00 00 02 00 -2 02 04 -2 00 00 -2
00 -2 00 02 02 -4 02 00 -2 00 02 00 04 02 00 02
00 00 00 00 00 00 00 00 -2 02 02 -2 02 -2 -2 -6
00 00 -2 -2 00 00 -2 -2 -4 00 -2 02 00 04 02 -2
00 04 -2 02 -4 00 02 -2 02 02 00 00 02 02 00 00
00 04 00 -4 04 00 04 00 00 00 00 00 00 00 00 00
00 -2 04 -2 -2 00 02 00 02 00 02 04 00 02 00 -2
00 02 02 00 -2 04 00 02 -4 -2 02 00 02 00 00 02
00 02 02 00 -2 -4 00 02 -2 00 00 -2 -4 02 -2 00
00 -2 -4 -2 -2 00 02 00 00 -2 04 -2 -2 00 02 00
```

- We consider the approximation involving S_{12}, S_{22}, S_{32}, and S_{34} as they have the highest bias according to our S-box.

S_{12}: $X_1 \oplus X_3 \oplus X_4 = Y_2$	with probability 12/16 and bias +1/4
S_{22}: $X_2 = Y_2 \oplus Y_4$	with probability 4/16 and bias −1/4
S_{32}: $X_2 = Y_2 \oplus Y_4$	with probability 4/16 and bias −1/4
S_{34}: $X_2 = Y_2 \oplus Y_4$	with probability 4/16 and bias −1/4

- According to the figure, using the linear approximation of the 1st round,

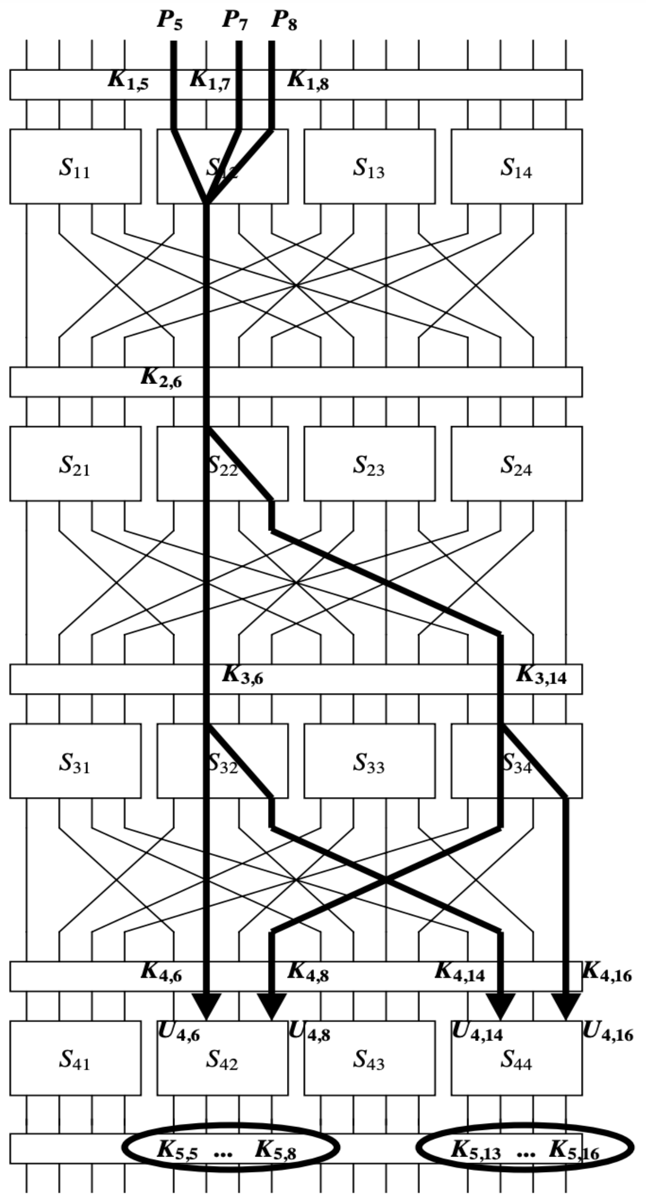
$$\begin{aligned} V_{1,6} &= U_{1,5} \oplus U_{1,7} \oplus U_{1,8} \\ &= (P_5 \oplus K_{1,5}) \oplus (P_7 \oplus K_{1,7}) \oplus (P_8 \oplus K_{1,8}) \end{aligned}$$ with a probability of 3/4

- For the approximation of the 2nd round,

$$V_{2,6} \oplus V_{2,8} = U_{2,6}$$ with a probability of ¼

- On combining both the equations, we get,

$$V_{2,6} \oplus V_{2,8} \oplus P_5 \oplus P_7 \oplus P_8 \oplus K_{1,5} \oplus K_{1,7} \oplus K_{1,8} \oplus K_{2,6} = 0$$ with probability 3/8 (bias of 1/8) by applying the Piling-Up Lemma



- For round 3, we get,

  $V_{3,6} \oplus V_{3,8} = U_{3,6}$, with a probability of 1/4

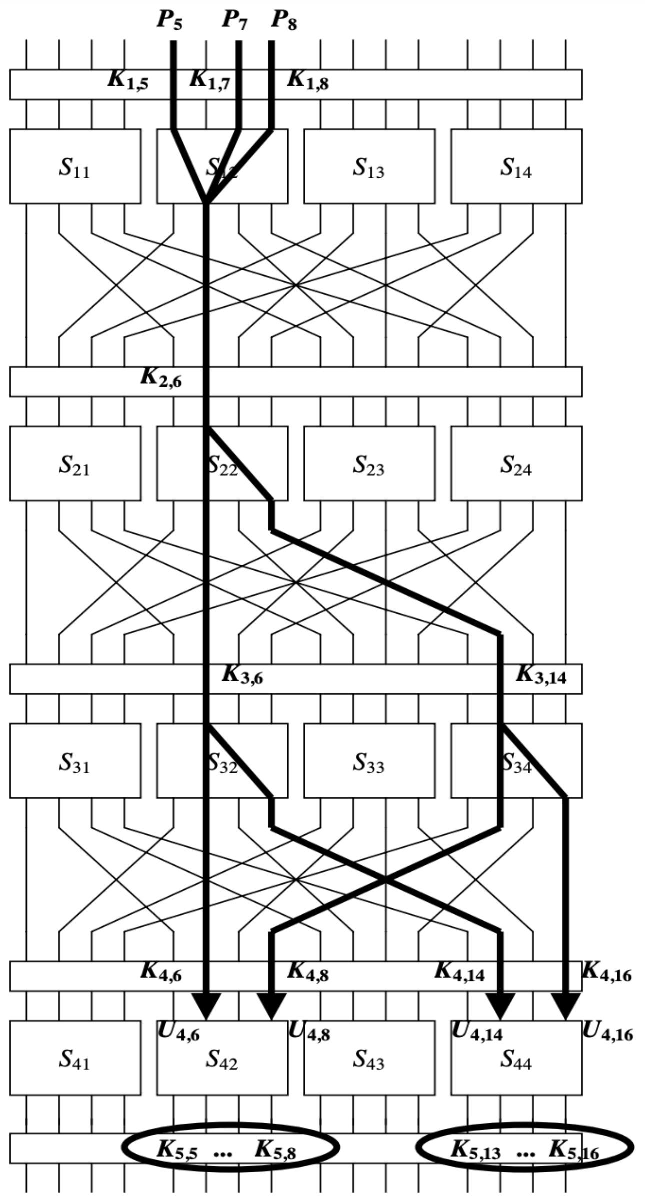
- and $V_{3,14} \oplus V_{3,16} = U_{3,14}$, with a probability of 1/4
- Then, by combining the equations, we get,

$V_{3,6} \oplus V_{3,8} \oplus V_{3,14} \oplus V_{3,16} \oplus V_{2,6} \oplus K_{3,6} \oplus V_{2,8} \oplus K_{3,14} = 0$, with a probability of 5/8 as computed by Piling-Up Lemma

- After resolving the equations, we get

$$U_{4,6} \oplus U_{4,8} \oplus U_{4,14} \oplus U_{4,16} \oplus P_5 \oplus P_7 \oplus P_8 = 0$$

  which must hold a probability of 15/32 or 17/32 (i.e magnitude of the bias must be 1/32)



Extracting Key Bits

- Once an R-1 round linear approximation is discovered for a cipher of R rounds with a suitably large enough linear probability bias, it is conceivable to attack the cipher by recovering bits of the last subkey.
- In our example, it is possible to extract bits from subkey K^5 given a 3 round linear approximation. The bits to be recovered from the last subkey are referred to as *target partial key*.
- Extracting the key bits involves the following process:
  - ❖ For all possible values of the target partial subkey, the ciphertext bits are XORed with the bits of the target partial subkey and the result is run backwards through the corresponding S-boxes.
  - ❖ This is done for all known plaintext/ciphertext samples and a count is kept for each value of the target partial subkey.
  - ❖ The count for a particular target partial subkey value is incremented when the linear expression holds true for the bits into the last round's S-boxes (determined by the partial decryption) and the known plaintext bits.
  - ❖ The target partial subkey value which has the count which differs the greatest from half the number of plaintext/ciphertext samples is assumed to represent the correct values of the target partial subkey bits.

- This works because it is assumed that the correct partial subkey value will result in the linear approximation holding with a probability significantly different from 1/2. (Whether it is above or below 1/2 depends on whether a linear or affine expression is the best approximation and this depends on the unknown values of the subkey bits implicitly involved in the linear expression.)
- In the context of our example, The linear expression

$$U_{4,6} \oplus U_{4,8} \oplus U_{4,14} \oplus U_{4,16} \oplus P_5 \oplus P_7 \oplus P_8 = 0$$

  affects the inputs to S-boxes S_{42} and S_{44} in the last round.
- For each plaintext/ciphertext sample, we would try all 256 values for the target partial subkey [$K_{5,5}$...$K_{5,8}$, $K_{5,13}$...$K_{5,16}$]. For each partial subkey value, we would increment the count whenever above equation holds true, where we determine the value of [$U_{4,5}$...$U_{4,8}$, $U_{4,13}$...$U_{4,16}$] by running the data backwards through the target partial subkey and S-boxes S_{24} and S_{44}. The count which deviates the largest from half of the number of plaintext/ciphertext samples is assumed to the correct value.

- We have simulated attacking our basic cipher by generating 10000 known plaintext/ciphertext values.
- the count which differed the most from 5000 corresponded to target partial subkey value [$K_{5,5}$...$K_{5,8}$, $K_{5,13}$...$K_{5,16}$] (as expected) confirming that the attack has successfully derived the subkey bits.

Contributions

- Mathematical Attacks on RSA (including codes and presentation slides) – Ishita Saraf
- SPN implementation and Linear cryptanalysis (including codes and presentation slides) – Anusha Aggarwal