Graph-based cryptography

Improved Perfect Code Cryptosystem 7

류지은	국민대
염용진	국민대
윤승태	CSHL
강주성	국민대
김용빈	국민대

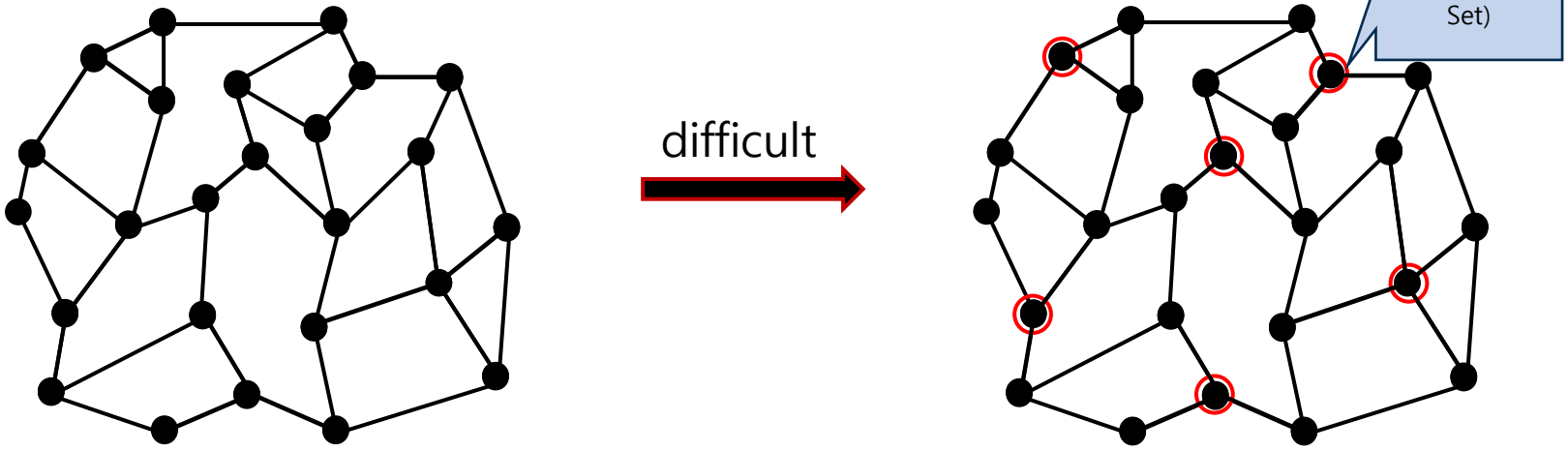
INDEX

Motivation

- In 1992, Koblitz and Fellows proposed a graph-based public key cryptosystem.
- The complexity of finding a certain subgraphs in a graph, and it is predicted to be an NP-hard problem that has potential applications in PQC.

Hard problems in Graph Theory

- To determine whether a given graph has a PDS is NP-hard.
- To find a PDS in a graph is believed to be infeasible.



M. Fellows, and N. Koblitz, "Combinatorially based cryptography for children (and adults)," Congressus Numerantium, 9

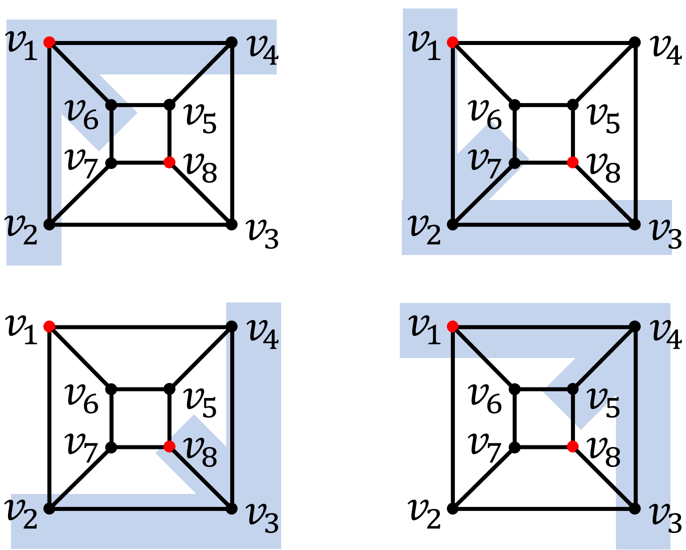
2. PDS & PDF in a graph

RASE KMU

PDS (Perfect Dominating Set)

For every vertex v in the graph $G = (V, E)$,
if $N[v]$ contains exactly one of the elements of vertex set $A \subseteq V$,
then A is called the PDS of G.

$$\exists A \subset V \ \text{s.t.} \ \forall v \in V, \qquad |N[v] \cap A| = 1$$



Every elements v of A belong
to the set $N[v]$ only once

PDS

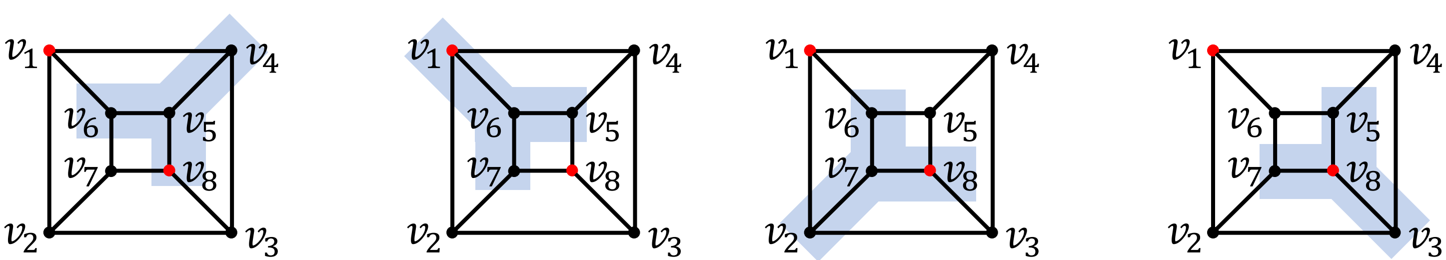
$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$ $\quad$ $A = \{v_1, v_8\}$

$N[v_1] = \{v_1, v_2, v_4, v_6\}$ $\quad$ $N[v_2] = \{v_1, v_2, v_3, v_7\}$

$N[v_3] = \{v_2, v_3, v_4, v_8\}$ $\quad$ $N[v_4] = \{v_1, v_3, v_4, v_5\}$

$N[v_5] = \{v_4, v_5, v_6, v_8\}$ $\quad$ $N[v_6] = \{v_1, v_2, v_4, v_6\}$

$N[v_7] = \{v_2, v_6, v_7, v_8\}$ $\quad$ $N[v_8] = \{v_3, v_5, v_7, v_8\}$



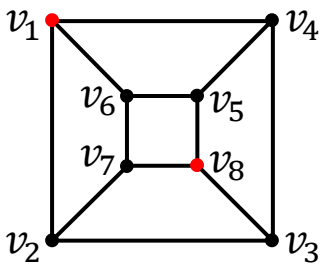
PDF (Perfect Dominating Function)

Let $f : V \to \{0,1\}$ is a function
that maps from vertex set V to set $\{0,1\}$ for graph $G = (V, E)$.
f is called a PDF if it is satisfying the following condition

$$\forall v \in V, \ \sum_{u \in N[v]} f(v) = 1$$

$$f(v) = x_v = \begin{cases} 1, if\ v \in A \\ 0, otherwise \end{cases}$$

assign each vertex of the PDS to 1
and all others to 0, then f is a PDF



0

$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$

$x_{v_1} = 1, \quad x_{v_2} = 0, \quad x_{v_3} = 0, \quad x_{v_4} = 0,$

$\sum_{u \in N[v_1]} x_u = x_{v_1} + x_{v_2} + x_{v_4} + x_{v_6} = 1$

$\sum_{u \in N[v_3]} x_u = x_{v_2} + x_{v_6} + x_{v_4} + x_{v_8} = 1$

$\sum_{u \in N[v_5]} x_u = x_{v_4} + x_{v_5} + x_{v_6} + x_{v_8} = 1$

$\sum_{u \in N[v_7]} x_u = x_{v_2} + x_{v_6} + x_{v_7} + x_{v_8} = 1$

1

$A = \{v_1, v_8\}$

$x_{v_5} = 0, \quad x_{v_6} = 0, \quad x_{v_7} = 0, \quad x_{v_8} = 1$

$\sum_{u \in N[v_2]} x_u = x_{v_1} + x_{v_2} + x_{v_3} + x_{v_7} = 1$

$\sum_{u \in N[v_4]} x_u = x_{v_1} + x_{v_3} + x_{v_4} + x_{v_5} = 1$

$\sum_{u \in N[v_6]} x_u = x_{v_1} + x_{v_2} + x_{v_4} + x_{v_6} = 1$

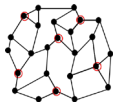
$\sum_{u \in N[v_8]} x_u = x_{v_3} + x_{v_5} + x_{v_7} + x_{v_8} = 1$

Invariant polynomial (of degree one)

Hard problem in Graph-based Public Key Cryptosytem

IPCC

public key: $G = (V, E)$ 

private key: PDS of G 

Message $M = m_1 + m_2 + \cdots + m_n$

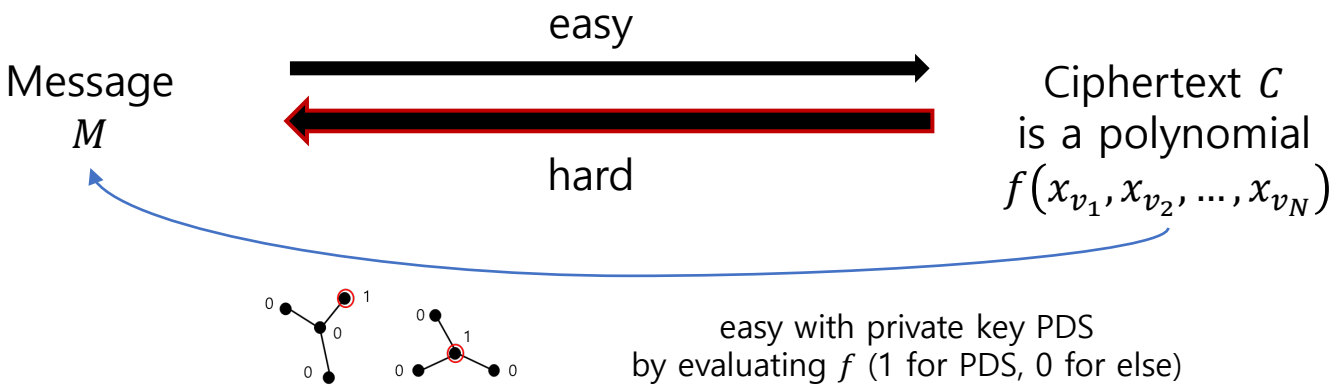
$m_1 p_{11}(\cdot)p_{12}(\cdot) + m_2 p_{21}(\cdot)p_{22}(\cdot) + \cdots + m_n p_{n1}(\cdot)p_{n2}(\cdot)$

$p_{ij}(\cdot)$: Invariant polynomials

Reduction

Ciphertext C

$C = f(x_{v_1}, x_{v_2}, \ldots, x_{v_N})$

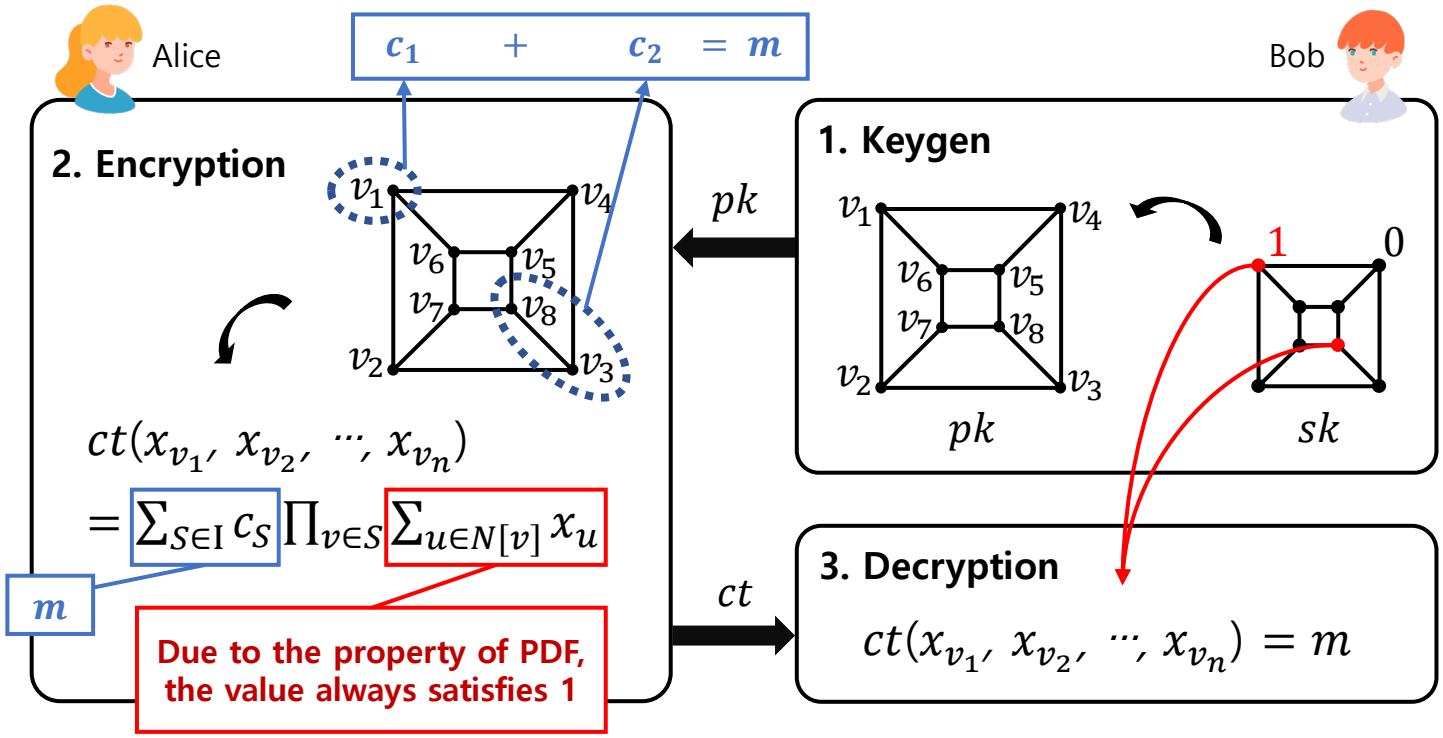


Perfect Code Cryptosystems

This cryptosystem relies on the problem of finding a PDS in the graph.

public key : 3-regular graph having PDSes

secret key : PDF $f(v) = x_v = \begin{cases} 1, & if\ v \in A\ (PDS) \\ 0, & otherwise \end{cases}$



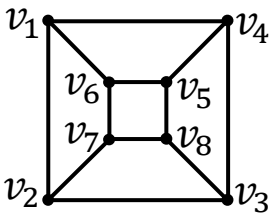
Improved Perfect Code Cryptosystems

public key : 3-regular graph having PDSes
secret key : PDF

Alice

Bob

2. Encryption



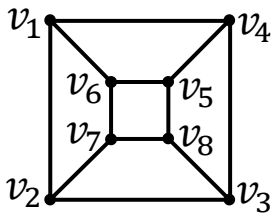
Invariant polynomial of degree $k_1 + k_2$

$$ct(x_{v_1}, x_{v_2}, \cdots, x_{v_n}) = F(m) = f_1(m_1) \times f_2(m_2)$$
$$= \sum_{S \in I} c_S \prod_{v \in S} \sum_{u \in N[v]} x_u \times \sum_{S \in I} c_S \prod_{v \in S} \sum_{u \in N[v]} x_u$$

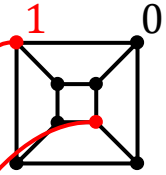
Invariant polynomial of degree k_1, k_2

pk

1. Keygen



pk



sk

ct

3. Decryption

$$ct(x_{v_1}, x_{v_2}, \cdots, x_{v_n}) = m_1 \times m_2$$

New attack technique and problem analysis for 1st round IPCC

Paragraphs about new attack technique among the feedback received on the proposed algorithm

> As an example, consider the first example in the KAT for the case of f1. This is given a message m = 18790. The ciphertext produced by the reference implementation contains the following list of coefficients (here stated without their multiplicities): [35, 9087, 14460, 16002, 16620, 21637, 22560, 24760, 33530, 36038, 36868, 38564, 39587, 39792, 62376].
> Summing these up gives us 411916 = 18790 mod 65521, which indeed is the plaintext. Note that the KAT file shows the hash of the ciphertext, not the ciphertext itself. We ran this attack on ciphertexts produced by the KAT. There are some few cases (2 out of 100 for f1, 0 out of 100 for f3, 8 out of 100 for f4) where this simple attack does not give the plaintext: in these cases, there are more than 15 coefficients, because variables repeated leading to combinations. We are still working on tracing through those to determine which of the coefficients we need to skip in summing up. We think that counting the frequency of occurrence will give us information. But we wanted to announce our findings so far as a fast attack with a success probability of more than 90% means that the system is typically broken.

To increase the attack complexity, **the degree k must be large**,

but the coefficients of high degree terms do not mix

Each coefficient generated in the encryption process is shared by 4^k terms

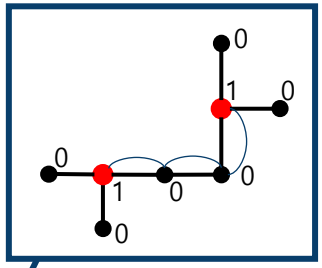
These two reasons appear to be the root of the problem

➡ The fundamental problem of the encryption process in the previous version

Concepts of Invariant polynomials in IPCC7

General invariant polynomial of degree k (recursively generated)

$\sum(\tilde{f}_l - a_l)\, x_{u_l}$ for message $\tilde{m}_j$ $(u_l \in N[u_j])$



Since $l = 0, 1, 2$ or 3 on 3-regular graph,

invariant polynomial: $(\tilde{f}_0 - a_0)x_{u_0} + (\tilde{f}_1 - a_1)x_{u_1} + (\tilde{f}_2 - a_2)x_{u_2} + (\tilde{f}_3 - a_3)x_{u_3}$

and $\tilde{f}_l$ is invariant polynomial of degree $\leq k - 1$

Due to the property PDF, only one coefficient $(\tilde{f}_X - a_X)$ for x_{u_X} s.t. $x_{u_X} = 1$ where $u_X \in N[u_j]$ will remain.

In the previous algorithm IPCC,

all $\tilde{f}_l - a_l$ were identical to the invariant polynomial for x_{u_l} of degree one

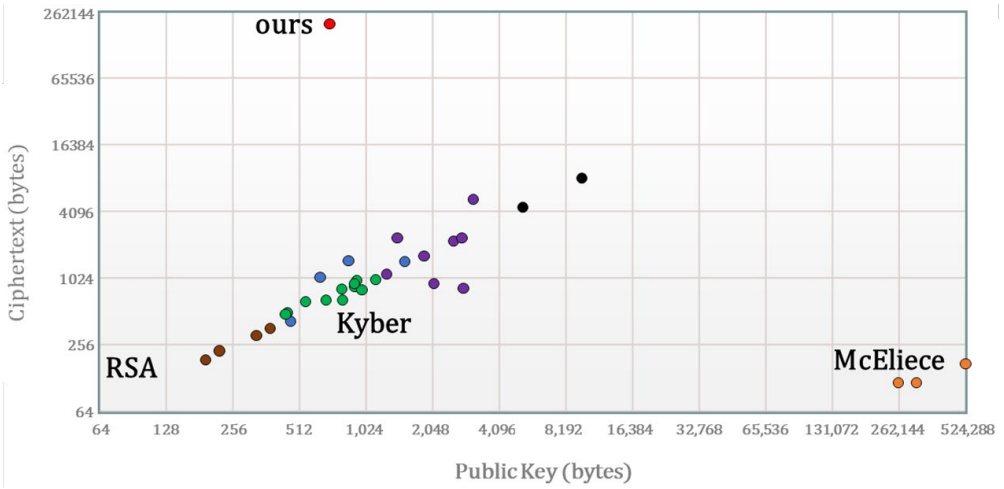
(ex. $\left(x_{u_0'} + x_{u_1'} + x_{u_2'} + x_{u_3'}\right)x_{u_0} + \left(x_{u_0'} + x_{u_1'} + x_{u_2'} + x_{u_3'}\right)x_{u_1} + \left(x_{u_0'} + x_{u_1'} + x_{u_2'} + x_{u_3'}\right)x_{u_2} + \left(x_{u_0'} + x_{u_1'} + x_{u_2'} + x_{u_3'}\right)x_{u_3}$
$= \left(x_{u_0'} + x_{u_1'} + x_{u_2'} + x_{u_3'}\right)(x_{u_0} + x_{u_1} + x_{u_2} + x_{u_3})$)

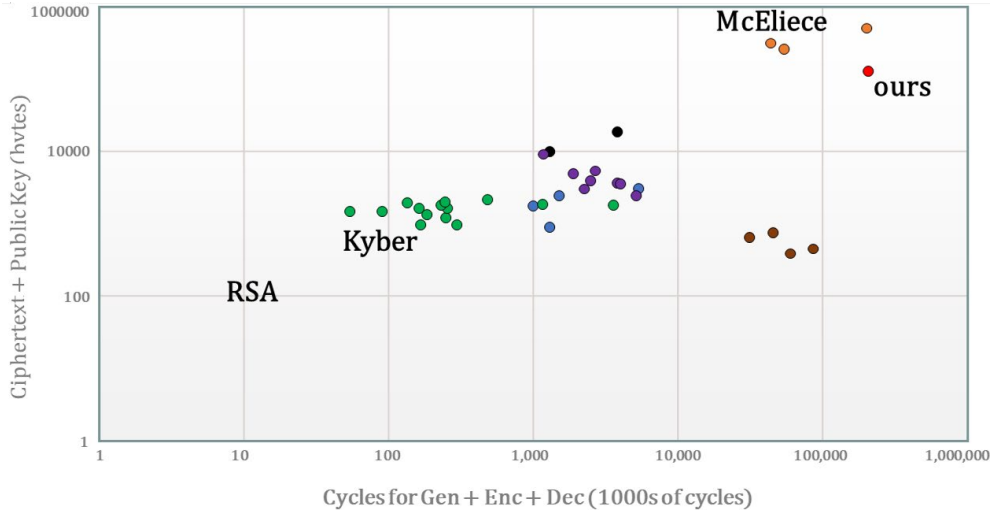
Comparison of IPCC and NIST PQC candidates

Features of IPCC

- Small public key(768 bytes)
  but huge ciphertext(250 KB)
- Fast decryption 2Gbps (intel I5)
  (naturally parallelizable)
- Suitable for whitebox cryptography
  (one-wayness) with large memory

$$
\begin{aligned}
ct = \quad & 485730577\, x_{v_{53}} x_{v_{119}} x_{v_{134}} x_{v_{169}} x_{v_{218}} x_{v_{229}} \\
& +1185214412\, x_{v_{16}} x_{v_{31}} x_{v_{38}} x_{v_{63}} x_{v_{93}} x_{v_{193}} x_{v_{220}} \\
& +1187133452\, x_{v_{53}} x_{v_{56}} x_{v_{120}} x_{v_{134}} x_{v_{169}} x_{v_{250}} x_{v_{255}} \\
& +\ 252775020\, x_{v_{53}} x_{v_{73}} x_{v_{164}} x_{v_{166}} x_{v_{206}} x_{v_{235}} x_{v_{250}} \\
& +1280749315\, x_{v_7} x_{v_{60}} x_{v_{83}} x_{v_{105}} x_{v_{191}} x_{v_{217}} \\
& +1393233314\, x_{v_{18}} x_{v_{53}} x_{v_{58}} x_{v_{85}} x_{v_{131}} x_{v_{206}} x_{v_{211}} \\
& +\ 832293056\, x_{v_{51}} x_{v_{60}} x_{v_{63}} x_{v_{196}} x_{v_{213}} x_{v_{247}} \\
& +\ 484242184\, x_{v_{63}} x_{v_{93}} x_{v_{108}} x_{v_{183}} x_{v_{206}} \\
& +\ \ 93293340\, x_{v_{31}} x_{v_{63}} x_{v_{65}} x_{v_{130}} x_{v_{146}} x_{v_{228}} x_{v_{229}} \\
& +1926696176\, x_{v_1} x_{v_{38}} x_{v_{53}} x_{v_{68}} x_{v_{134}} x_{v_{141}} \\
& +1442193836\, x_{v_{31}} x_{v_{63}} x_{v_{67}} x_{v_{124}} x_{v_{125}} x_{v_{166}} \\
& +2017528070\, x_{\cdots} x_{\cdots} x_{\cdots} x_{\cdots} x_{\cdots} x_{\cdots} x_{\cdots}
\end{aligned}
$$





6. Bernstein's attack on IPCC7

RASE KMU

IPCC7 key recovery

D. J. Bernstein djb@... 9월 29일 (금) 오후 8:11 (12일 전)
나, coji67, yoon, jskang, salt, Tanja, Jolijn에게

Dear designers of the IPCC system,

It appears to be possible to efficiently find secret keys from IPCC7 public keys with high probability. Please see the attached Sage script for details. Inserting

```
for (long long i = 0;i < 6*NUMPDS;++i)
  printf("pk %lld %d %d\n",i,(int) pk[i][0],(int) pk[i][1]);
printf("pkend\n");
```

into the C code for IPCC7 produces the appropriate input format for the Sage script. In experiments with 10 keys, 7 partitioned the vertices into four sets of size 64, and 3 partitioned the vertices into two sets of size 64, two sets of size 63, and two sets of size 1. Any of the sets of size 64 should work for decryption, and we checked that one example matches the original secret key.

We also noticed that the public keys include A-B edges followed by A-C edges, allowing a simpler attack that intersects those edges to find A. Sorting the edges before releasing public keys would stop this simpler attack, but would not affect the attached Sage script.

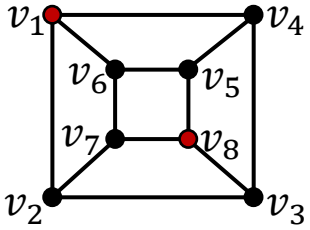
Please let us know if you have any questions.

All the best
Dan, Jolijn, and Tanja

```
numvertices = 256

neighbors = {}

def attack():
  M = []
  for v in neighbors:
    Mj = [0]*numvertices
    Mj[v] = 1
    for w in neighbors[v]:
      Mj[w] = 1
    M += [Mj]

  M = matrix(QQ,M)
  M = M.echelon_form()

  todo = []
  for row in M:
    for j in range(len(row)):
      if row[j] != 0:
        assert row[j] == 1
        todo += [(j,[(i,row[i]) for i in range(len(row)) if row[i] != 0])]
        break

  classification = {}
  for v in range(numvertices):
    c = vector(QQ,[i == v for i in range(numvertices)])
    for j,iri in todo:
      cj = c[j]
      if cj == 0: continue
      for i,ri in iri:
        c[i] -= cj*ri
    c.set_immutable()
    classification[v] = c

  for c in set(classification[j] for j in classification):
    part = [j for j in sorted(classification) if classification[j] == c]
    print(len(part),'element(s):',' '.join(str(j) for j in part))
  print('')

for line in sys.stdin:
  line = line.split()
  if line[0] == 'pk':
    v0,v1 = map(int,line[2:4])
    if v0 not in neighbors:
      neighbors[v0] = []
    neighbors[v0] += [v1]
    if v1 not in neighbors:
      neighbors[v1] = []
    neighbors[v1] += [v0]
  if line[0] == 'pkend':
    attack()
    neighbors = {}
```

6. RREF attack on IPCC7

$$\begin{pmatrix} 1&1&0&1&0&1&0&0\\ 1&1&1&0&0&0&1&0\\ 0&1&1&1&0&0&0&1\\ 1&0&1&1&1&0&0&0\\ 0&0&0&1&1&1&0&1\\ 1&0&0&0&1&1&1&0\\ 0&1&0&0&0&1&1&1\\ 0&0&1&0&1&0&1&1 \end{pmatrix}$$



Example graph

Gaussian elimination

$$\begin{pmatrix} 1&0&0&0&0&0&0&-1\\ 0&1&0&0&0&1&0&1\\ 0&0&1&0&0&-1&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&1&0&1\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0 \end{pmatrix}$$

todo = [(1, [(1, 1), (8, -1)]),
(2, [(2, 1), (6, 1), (8, 1)]),
(3, [(3, 1), (6, -1)]),
(4, [(4, 1)]),
(5, [(5, 1), (6, 1), (8, 1)]),
(6, [(7, 1)])]

$I - \mathrm{RREF}(M)$

$$\begin{pmatrix} 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&-1&0&-1\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&-1&0&-1\\ 0&0&0&0&0&1&-1&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{pmatrix}$$

$\{v_2, v_5\}$

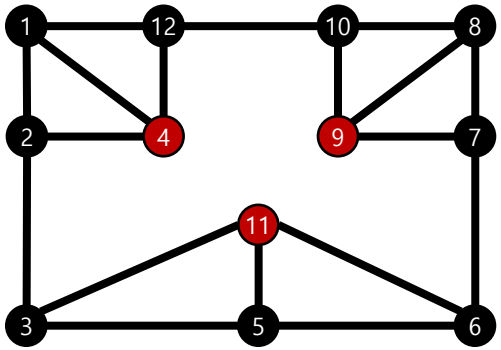
$\{v_1, v_8\}$

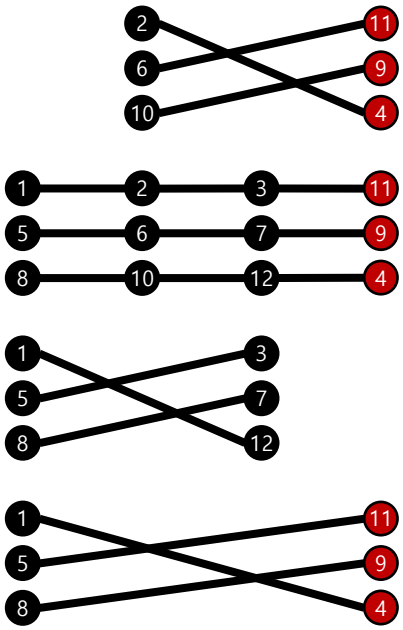
```
numvertices = 256

neighbors = {}

def attack():
  M = []
  for v in neighbors:
    Mj = [0]*numvertices
    Mj[v] = 1
    for w in neighbors[v]:
      Mj[w] = 1
    M += [Mj]

  M = matrix(QQ,M)
  M = M.echelon_form()

  todo = []
  for row in M:
    for j in range(len(row)):
      if row[j] != 0:
        assert row[j] == 1
        todo += [(j,[(i,row[i]) for i in range(len(row)) if row[i] != 0])]
        break

  classification = {}
  for v in range(numvertices):
    c = vector(QQ,[i == v for i in range(numvertices)])
    for j,iri in todo:
      cj = c[j]
      if cj == 0: continue
      for i,ri in iri:
        c[i] -= cj*ri
    c.set_immutable()
    classification[v] = c

  for c in set(classification[j] for j in classification):
    part = [j for j in sorted(classification) if classification[j] == c]
    print(len(part),'element(s):',' '.join(str(j) for j in part))
  print('')

for line in sys.stdin:
  line = line.split()
  if line[0] == 'pk':
    v0,v1 = map(int,line[2:4])
    if v0 not in neighbors:
      neighbors[v0] = []
    neighbors[v0] += [v1]
    if v1 not in neighbors:
      neighbors[v1] = []
    neighbors[v1] += [v0]
  if line[0] == 'pkend':
    attack()
    neighbors = {}
```

6. 3-regular graph generated in KeyGen of IPCC7

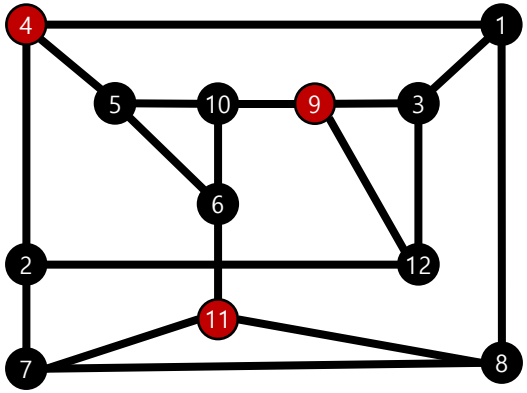
case1

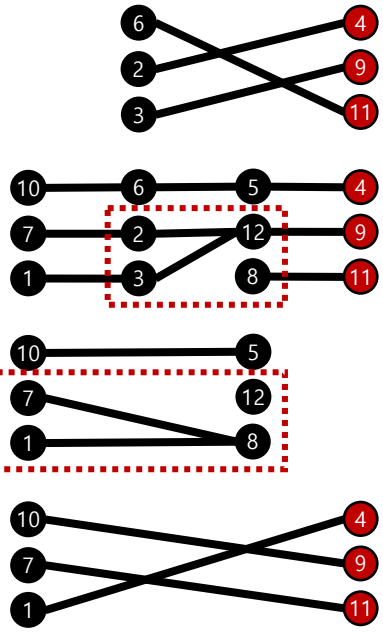
KeyGen in IPCC7



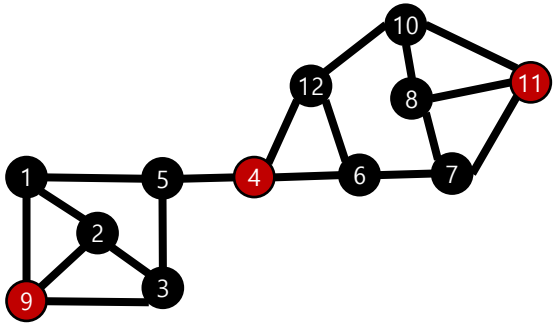


case2





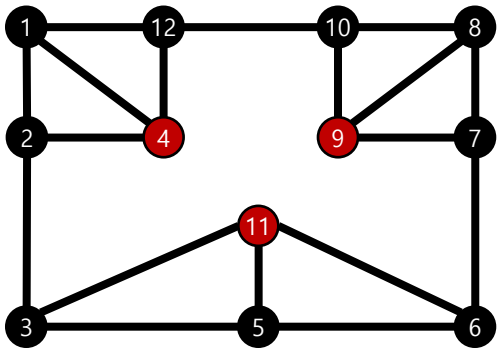
case3

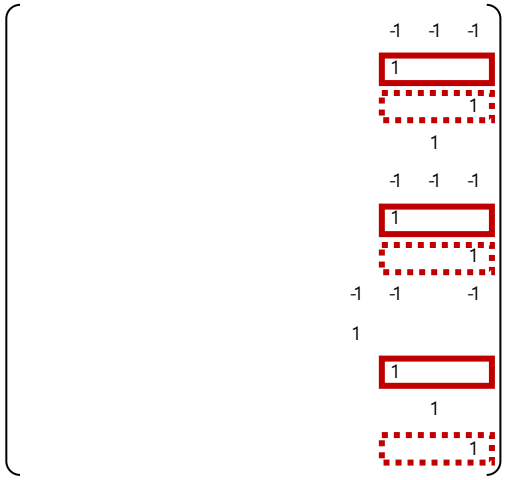


New type of 3-regular graph having a PDS

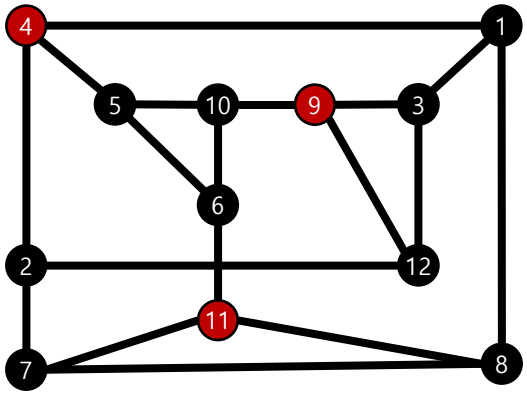
7. 3-regular graph secure against Bernstein's RREF attack

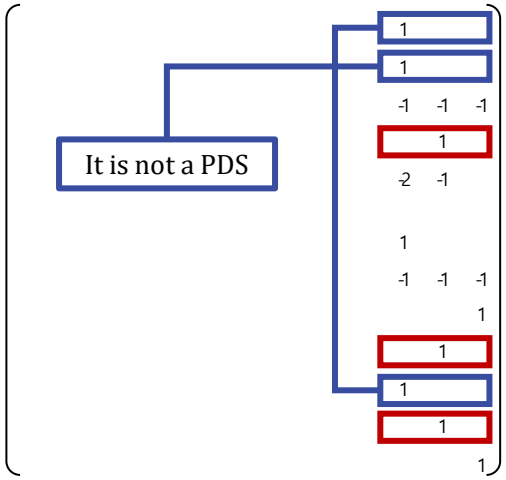
case1



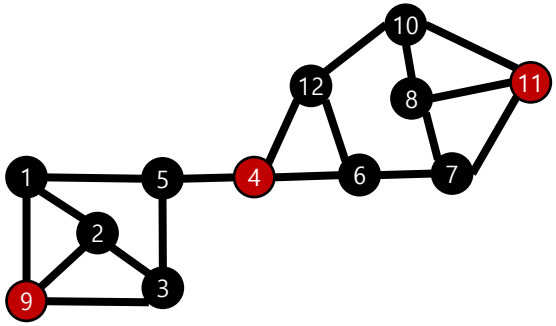


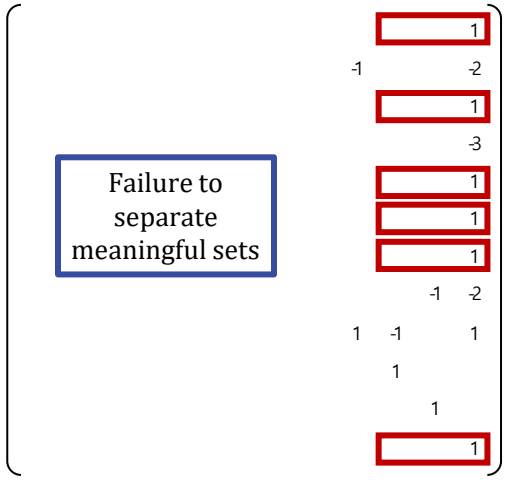
case2





case3





Improved Perfect Code Cryptosystem

감사합니다