Application of the new modeling technique on Trivium-like ciphers

No Author Given

No Institute Given

1 Specification of Trivium-like ciphers

1.1 Trivium

Trivium [1] is a stream cipher based on the nonlinear feedback shift register (NFSR) designed by De Cannière and Preneel, which has a 288-bit internal state $(s_0, s_1, \ldots, s_{287})$ distributed on three Registers A, B and C. The 80-bit secret key $\boldsymbol{k} = (k_0, \ldots, k_{79})$ and 80-bit IV $\boldsymbol{v} = (v_0, \ldots, v_{79})$ are loaded into the Registers A and B respectively. The other state bits are set to 0 except the last three bits in the Register C. That is, we have

$$\begin{aligned}(s_0, s_1, \ldots, s_{92}) &\leftarrow (k_0, k_1, \ldots, k_{79}, 0, \ldots, 0),\\ (s_{93}, s_{94}, \ldots, s_{176}) &\leftarrow (v_0, v_1, \ldots, v_{79}, 0, \ldots, 0),\\ (s_{177}, s_{178}, \ldots, s_{287}) &\leftarrow (0, 0, \ldots, 0, 1, 1, 1).\end{aligned}$$

The pseudo-code of the update function is given as follows.

$$\begin{aligned}(t_1, t_2, t_3) &\leftarrow (s_{65} \oplus s_{92}, s_{161} \oplus s_{176}, s_{242} \oplus s_{287}),\\ z &\leftarrow t_1 \oplus t_2 \oplus t_3,\\ (s_0, s_1, \ldots, s_{92}) &\leftarrow (t_1 \oplus s_{90} \cdot s_{91} \oplus s_{170}, s_0, \ldots, s_{91}),\\ (s_{93}, s_{94}, \ldots, s_{176}) &\leftarrow (t_2 \oplus s_{174} \cdot s_{175} \oplus s_{263}, s_{93}, \ldots, s_{175}),\\ (s_{177}, s_{178}, \ldots, s_{287}) &\leftarrow (t_3 \oplus s_{285} \cdot s_{286} \oplus s_{68}, s_{177}, \ldots, s_{286}),\end{aligned}$$

where z denotes the 1-bit key stream. During the initialization phase, the state is updated 4×288 rounds without producing an output. After the initialization, the 1-bit key stream is produced by every update function.

1.2 Kreyvium

Kreyvium [2] is designed for the use of fully Homomorphic encryption and is a variant of Trivium with 128-bit security. Compared with Trivium, Kreyvium uses two extra registers (K^*, V^*) without updating but shifting and adds a single bit of (K^*, V^*) to the update functions of Registers A and C, where K^* and V^* only involve the secret key bits and public IV bits respectively. Trivium

uses an 80-bit key and an 80-bit IV, while Kreyvium uses a 128-bit key and a 128-bit IV. The registers are initialized as

$$\begin{aligned}(s_0, s_1, \ldots, s_{92}) &\leftarrow (k_0, k_1, \ldots, k_{92}),\\ (s_{93}, s_{94}, \ldots, s_{176}) &\leftarrow (v_0, v_1, \ldots, v_{83}),\\ (s_{177}, s_{178}, \ldots, s_{287}) &\leftarrow (v_{84}, v_{85}, \ldots, v_{127}, 1, \ldots, 1, 0),\\ (V_0, V_1, \ldots, V_{127}) &\leftarrow (v_{127}, v_{126}, \ldots, v_0),\\ (K_0, K_1, \ldots, K_{127}) &\leftarrow (k_{127}, v_{126}, \ldots, k_0).\end{aligned}$$

The pseudo-code of the update function is given as follows.

$$\begin{aligned}(t_1, t_2, t_3) &\leftarrow (s_{65} \oplus s_{92}, s_{161} \oplus s_{176}, s_{242} \oplus s_{287} \oplus K_0)\\ z &\leftarrow t_1 \oplus t_2 \oplus t_3,\\ (s_0, s_1, \ldots, s_{92}) &\leftarrow (t_1 \oplus s_{90} \cdot s_{91} \oplus s_{170} \oplus V_0, s_0, \ldots, s_{91}),\\ (s_{93}, s_{94}, \ldots, s_{176}) &\leftarrow (t_2 \oplus s_{174} \cdot s_{175} \oplus s_{263}, s_{93}, \ldots, s_{175}),\\ (s_{177}, s_{178}, \ldots, s_{287}) &\leftarrow (t_3 \oplus s_{285} \cdot s_{286} \oplus s_{68}, s_{177}, \ldots, s_{286}),\\ (K_{127}, K_{126}, \ldots, K_0) &\leftarrow (K_0, K_{127}, \ldots, K_1),\\ (V_{127}, V_{126}, \ldots, V_0) &\leftarrow (V_0, V_{127}, \ldots, V_1).\end{aligned}$$

where z denotes the 1-bit key stream. Like Trivium, the number of initialization rounds of Kreyvium is 1152.

2 The propagation rules of 3SDP/u for basic Boolean functions

2.1 The propagation rules for three basic functions.

Proposition 1 (COPY). *Let $\boldsymbol{x} = (x_0, x_1, ..., x_{m-1})$ and $\boldsymbol{y} = (x_0, x_0, x_1, ..., x_{m-1})$ be the input and output vector of a Copy function. Let $\mathbb{X}$ and $\mathbb{Y}$ be the input and output multisets, respectively. Assuming that $\mathbb{X}$ has $\mathcal{T}_{\mathbb{L}}^{1^m}$, $\mathbb{Y}$ has $\mathcal{T}_{\mathbb{L}'}^{1^{m+1}}$ where $\mathbb{L}'$ is computed as*

$$\mathbb{L}' \leftarrow (l_0', l_0'', l_1, \ldots, l_{m-1}), \text{ if } l_0' \vee l_0'' = l_0,$$

for all $\boldsymbol{l} \in \mathbb{L}$. Here $\mathbb{L}' \leftarrow \boldsymbol{l}$ denotes that $\boldsymbol{l}$ is inserted into the multiset $\mathbb{L}'$.

Proposition 2 (AND). *Let $\boldsymbol{x} = (x_1, x_2, ..., x_m)$ and $\boldsymbol{y} = (x_1 \cdot x_2, x_3, ..., x_m)$ be the input and output vector of an And function. Let $\mathbb{X}$ and $\mathbb{Y}$ be the input and output multisets, respectively. Assuming that $\mathbb{X}$ has $\mathcal{T}_{\mathbb{L}}^{1^m}$, $\mathbb{Y}$ has $\mathcal{T}_{\mathbb{L}'}^{1^{m-1}}$ where $\mathbb{L}'$ is computed as*

$$\mathbb{L}' \leftarrow (l_0, l_2, \ldots, l_{m-1}), \text{ if } l_0 = l_1,$$

for all $\boldsymbol{l} \in \mathbb{L}$.

Proposition 3 (XOR). *Let $\boldsymbol{x} = (x_1, x_2, ..., x_m)$ and $\boldsymbol{y} = (x_1 \oplus x_2, x_3, ..., x_m)$ be the input and output vector of a Xor function. Let $\mathbb{X}$ and $\mathbb{Y}$ be the input and*

output multisets, respectively. Assuming that $\mathbb{X}$ *has* $\mathcal{T}_{\mathbb{L}}^{1^m}$, $\mathbb{Y}$ *has* $\mathcal{T}_{\mathbb{L}'}^{1^{m-1}}$ *where* $\mathbb{L}'$ *is computed as*

$$\mathbb{L}' \leftarrow (l_0 + l_1, l_2, \ldots, l_{m-1}), \text{ if } l_0 \cdot l_1 = 0,$$

for all $\boldsymbol{l} \in \mathbb{L}$.

2.2 MILP models for three basic functions.

We introduce the MILP models for the three basic functions COPY, AND, and XOR. Note that these models are first proposed in [3,4], and describe the generalized forms of their propagation rules, which are easily deduced from their original forms. When constructing the MILP model, we usually initialize an empty model $\mathcal{M}$, then generate some MILP variables $v_1, \ldots, v_m$ by $\mathcal{M}.var \leftarrow v_1, \ldots, v_m$, which are used in the inequalities. Next, the inequality is added to the model $\mathcal{M}.con \leftarrow v_2 + \cdots + v_m \geq v_1$. Finally, we can call an off-the-shelf MILP solver to solve the model and obtain the results, including feasible or infeasible. In this paper, the MILP tool we used is the Gurobi optimizer.

Proposition 4 (MILP model for COPY). *Let* $a \xrightarrow{COPY} (b_0, b_1, \ldots, b_{m-1})$ *be a three-subset division trail of* ***COPY****. The following inequalities are sufficient to describe the propagation of the modified three-subset division property for* ***COPY****.*

$$\begin{cases} \mathcal{M}.var \leftarrow a, b_0, \ldots, b_{m-1} \text{ as binary;} \\ \mathcal{M}.con \leftarrow a = b_0 \vee b_1 \vee \cdots \vee b_{m-1}. \end{cases}$$

Note that the Gurobi optimizer supports the Or $(\vee)$ *operation.*

Proposition 5 (MILP model for AND). *Let* $(a_0, a_1, \ldots, a_{m-1}) \xrightarrow{AND} b$ *be a three-subset division trail of* ***AND****. The following inequalities are sufficient to describe the propagation of the modified three-subset division property for* ***AND****.*

$$\begin{cases} \mathcal{M}.var \leftarrow a_0, \ldots, a_{m-1}, b \text{ as binary;} \\ \mathcal{M}.con \leftarrow b = a_i, \forall i \in \{0, 1, \ldots, m-1\}. \end{cases}$$

Proposition 6 (MILP model for XOR). *Let* $(a_0, a_1, \ldots, a_{m-1}) \xrightarrow{XOR} b$ *be a three-subset division trail of* ***XOR****. The following inequalities are sufficient to describe the propagation of the modified three-subset division property for* ***XOR****.*

$$\begin{cases} \mathcal{M}.var \leftarrow a_0, \ldots, a_{m-1}, b \text{ as binary;} \\ \mathcal{M}.con \leftarrow b = a_0 + a_1 + \cdots + a_{m-1}. \end{cases}$$

3 The new MILP model $\mathcal{M}_{64}$ for Trivium-like ciphers

3.1 Expression Construction

Trivium and Kreyvium in Algorithm 1 is used to compute the ANFs of the state bits with respect to $\boldsymbol{k}$ and $\boldsymbol{v}$. Constructing the expression Eq.(1) derived by r-round Trivium-like cipher $f(\boldsymbol{k}, \boldsymbol{v})$ is illustrated as ParamSelector64 in Algorithm 1.

Algorithm 1: The Auxiliary Functions for Building an MILP Model

procedure TRIVIUM(state $\boldsymbol{s}$, cube indices I, round R)
  $\boldsymbol{c} = (c_0, \ldots, c_{79}) \in \mathbb{F}_2^{80}$, if $i \in I$, $c_i = 1$, otherwise $c_i = 0$
  $(s_0, s_1, \ldots, s_{92}) \leftarrow (k_0, \ldots, k_{79}, 0, \ldots, 0)$
  $(s_{93}, s_{94}, \ldots, s_{176}) \leftarrow (c_0 \cdot v_0, \ldots, c_{79} \cdot v_{79}, 0, \ldots, 0)$
  $(s_{177}, s_{178}, \ldots, s_{287}) \leftarrow (0, \ldots, 0, 1, 1, 1)$
  for i **from** 1 **to** R **do**
    $t_1 \leftarrow s_{65} \oplus s_{90} \cdot s_{91} \oplus s_{92} \oplus s_{170}$
    $t_2 \leftarrow s_{161} \oplus s_{174} \cdot s_{175} \oplus s_{176} \oplus s_{263}$
    $t_3 \leftarrow s_{242} \oplus s_{285} \cdot s_{286} \oplus s_{287} \oplus s_{68}$
    $\boldsymbol{s} \leftarrow (t_3, s_0, \ldots, s_{91}, t_1, s_{93}, \ldots, s_{175}, t_2, s_{177}, \ldots, s_{286})$
  Return $\boldsymbol{s}$
end procedure

procedure KREYVIUM(state $\boldsymbol{s}$, cube indices I, round R)
  $\boldsymbol{c} = (c_0, \ldots, c_{127}) \in \mathbb{F}_2^{128}$, if $i \in I$, $c_i = 1$, otherwise $c_i = 0$
  $(s_0, s_1, \ldots, s_{92}) \leftarrow (k_0, k_1, \ldots, k_{92})$
  $(s_{93}, s_{94}, \ldots, s_{176}) \leftarrow (c_0 \cdot v_0, \ldots, c_{83} \cdot v_{83})$
  $(s_{177}, s_{178}, \ldots, s_{287}) \leftarrow (c_{84} \cdot v_{84}, \ldots, c_{127} \cdot v_{127}, 1, \ldots, 1, 0)$
  $(V_{127}, V_{126}, \ldots, V_0) \leftarrow (c_0 \cdot v_0, c_1 \cdot v_1, \ldots, c_{127} \cdot v_{127})$
  $(K_{127}, K_{126}, \ldots, K_0) \leftarrow (k_0, k_1, \ldots, k_{127})$
  for i **from** 1 **to** R **do**
    $t_1 \leftarrow s_{65} \oplus s_{90} \cdot s_{91} \oplus s_{92} \oplus s_{170} \oplus V_{(i-1) \mod 128}$
    $t_2 \leftarrow s_{161} \oplus s_{174} \cdot s_{175} \oplus s_{176} \oplus s_{263}$
    $t_3 \leftarrow s_{242} \oplus s_{285} \cdot s_{286} \oplus s_{287} \oplus s_{68} \oplus K_{(i-1) \mod 128}$
    $\boldsymbol{s} \leftarrow (t_3, s_0, \ldots, s_{91}, t_1, s_{93}, \ldots, s_{175}, t_2, s_{177}, \ldots, s_{286})$
  Return $\boldsymbol{s}$
end procedure

procedure PARAMSELECTOR64(cube indices I)
  $s_i \in \mathbb{F}_2[\boldsymbol{k}, \boldsymbol{v}]$ for $i \in \{0, \ldots, 287\}$ ▷ $\boldsymbol{k} \in \mathbb{F}_2^n$, $\boldsymbol{v} \in \mathbb{F}_2^m$
  $\boldsymbol{s} \leftarrow$ TRIVIUM($\boldsymbol{s}, I, 64$) (or KREYVIUM($\boldsymbol{s}, I, 64$))
  Prepare an empty polynomial sequence $\boldsymbol{g}^{(k)}$
  for i **from** 0 **to** $n-1$ **do**
    $g_i^{(k)} \leftarrow k_i$
  $loc \leftarrow n$
  for i **from** 0 **to** 287 **do**
    Decompose $s_i = h_{i,1}(\boldsymbol{k}, \boldsymbol{v}) \oplus h_{i,2}(\boldsymbol{k})$ ▷ there is no monomial $\boldsymbol{k}^{\boldsymbol{\mu}}$ in $h_{i,1}(\boldsymbol{k}, \boldsymbol{v})$ for any $\boldsymbol{\mu} \in \mathbb{F}_2^n$
    if there are at least two monomials in $h_{i,2}(\boldsymbol{k})$ **then**
      if $h_{i,2}(\boldsymbol{k})$ is included in $\boldsymbol{g}^{(k)}$ **and** $g_j^{(k)} = h_{i,2}(\boldsymbol{k})$ **then**
        $s_i \leftarrow h_{i,1}(\boldsymbol{k}, \boldsymbol{v}) \oplus k_j$
      else
        $g_{loc}^{(k)} \leftarrow h_{i,2}(\boldsymbol{k})$
        $s_i \leftarrow h_{i,1}(\boldsymbol{k}, \boldsymbol{v}) \oplus k_{loc}$
        $loc \leftarrow loc + 1$
  Return $\boldsymbol{s}$, $\boldsymbol{g}^{(k)}$
end procedure

3.2 The new MILP model of Trivium

TRIVIUMINIT64 in Algorithm 2 is used to model the algebraic relation $\boldsymbol{s}_{64} = \boldsymbol{E}'_{64}(\boldsymbol{x}, \boldsymbol{v})$ in the new composite function, i.e.,

$$f(\boldsymbol{x}, \boldsymbol{v}) \triangleq F_{out} \circ \boldsymbol{F}_r \circ \cdots \circ \boldsymbol{F}_{65} \circ \boldsymbol{E}'_{64}(\boldsymbol{x}, \boldsymbol{v}), \tag{1}$$

where $\boldsymbol{s}_{64} = \boldsymbol{E}'_{64}(\boldsymbol{x}, \boldsymbol{v}) = (s_{64,0}(\boldsymbol{x}, \boldsymbol{v}), \ldots, s_{64,287}(\boldsymbol{x}, \boldsymbol{v}))$. It is worth noting that in TRIVIUMINIT64, since some secret key variables are not involved in the ANF of any state bit $s_{64,i}(\boldsymbol{x}, \boldsymbol{v})$ for $i \in \{0, \ldots, 287\}$, we set these variables directly to 0-constant to avoid existing extra three-subset division trails. As the traditional

modeling technique, for a given set of cube indices I, we set the non-cube variables to 0-constant. Modeling the ANF of the state bit with respect to $\boldsymbol{k}$ and $\boldsymbol{v}$ is illustrated as ModelPoly in Algorithm 2.

3.3 The new MILP model of Kreyvium

KreyviumInit64 in Algorithm 3 is used to model the algebraic relation $\boldsymbol{s}_{64} = \boldsymbol{E}'_{64}(\boldsymbol{x}, \boldsymbol{v})$ in Eq.(1). It is worth noting that in KreyviumInit64 since some secret key variables are not involved in the ANF of any state bit $s_{64,i}(\boldsymbol{x}, \boldsymbol{v})$ for $i \in \{0, \ldots, 287\}$, we set these variables directly to 0-constant to avoid existing extra three-subset division trails. As the traditional modeling technique, for a given set of cube indices I, we set the non-cube variables to 0-constant.

Algorithm 2: The New MILP Model for Trivium

procedure TriviumModel(Round R, I, Monomial $\boldsymbol{\omega}$)
  $\boldsymbol{Ps}$, $\boldsymbol{g}^{(k)} \leftarrow$ ParamSelector64(I)
  $n \leftarrow \boldsymbol{g}^{(k)}.size()$ ▷ Assign the number of the secret key variables to n
  Prepare an empty MILP Model $\mathcal{M}_{64}$
  $\mathcal{M}_{64}.var \leftarrow \boldsymbol{s} = (s_0, \ldots, s_{287})$ as binary variables
  $\mathcal{M}_{64}.var \leftarrow \boldsymbol{k} = (k_0, \ldots, k_{n-1})$ as binary variables
  $\mathcal{M}_{64}.var \leftarrow \boldsymbol{v} = (v_0, \ldots, v_{79})$ as binary variables
  $\boldsymbol{s} \leftarrow$ TriviumInit64($\mathcal{M}_{64}, \boldsymbol{Ps}, \boldsymbol{s}, \boldsymbol{k}, \boldsymbol{v}, n$)
  for r **from** 65 **to** R **do**
    TriviumCore($\mathcal{M}_{64}, \boldsymbol{s}, 65, 170, 90, 91, 92$)
    TriviumCore($\mathcal{M}_{64}, \boldsymbol{s}, 161, 263, 174, 175, 176$)
    TriviumCore($\mathcal{M}_{64}, \boldsymbol{s}, 242, 68, 285, 286, 287$)
    $\boldsymbol{s} \leftarrow (s_{287}, s_0, \ldots, s_{286})$
  if $\boldsymbol{\omega} = (0, \ldots, 0)$ **then**
    $\mathcal{M}_{64}.con \leftarrow s_i = 0,\ i \in \{0, \ldots, 287\} \setminus \{65, 92, 161, 176, 242, 287\}$
    $\mathcal{M}_{64}.con \leftarrow s_{65} + s_{92} + s_{161} + s_{176} + s_{242} + s_{287} = 1$
  else
    $\mathcal{M}_{64}.con \leftarrow s_i = \omega_i,\ i \in \{0, \ldots, 287\}$
  return $\mathcal{M}_{64}, \boldsymbol{g}^{(k)}$
end procedure

procedure TriviumInit64($\mathcal{M}, \boldsymbol{Ps}, \boldsymbol{s}, \boldsymbol{k}, \boldsymbol{v}, n$)
  $\boldsymbol{n}^k, \boldsymbol{n}^v \leftarrow$ **CountNum**($\boldsymbol{Ps}$) ▷ Count the number of $\boldsymbol{k}$ and $\boldsymbol{v}$ in the polynomials $\boldsymbol{Ps}$
  for i **from** 0 **to** $n-1$ **do**
    if $n_i^{(k)} = 0$ **then**
      $\mathcal{M}.con \leftarrow k_i = 0$
    else
      $\mathcal{M}.var \leftarrow k'_i = (k'_{i,1}, k'_{i,2}, \ldots, k'_{i,n_i^{(k)}})$ as binary variables
      $\mathcal{M}.con \leftarrow k_i = k'_{i,1} \vee \cdots \vee k'_{i,n_i^{(k)}}$
  for i **from** 0 **to** 79 **do**
    if $n_i^{(v)} = 0$ **then**
      $\mathcal{M}.con \leftarrow v_i = 0$
    else
      $\mathcal{M}.con \leftarrow v_i = 1$
      $\mathcal{M}.var \leftarrow v'_i = (v'_{i,1}, v'_{i,2}, \ldots, v'_{i,n_i^{(v)}})$ as binary variables
      $\mathcal{M}.con \leftarrow v_i = v'_{i,1} \vee \cdots \vee v'_{i,n_i^{(v)}}$
  $\boldsymbol{c}^k \leftarrow (1, \ldots, 1) \in \mathbb{Z}^n$, $\boldsymbol{c}^v \leftarrow (1, \ldots, 1) \in \mathbb{Z}^{80}$
  $\boldsymbol{k'} \leftarrow (k'_0, \ldots, k'_{n-1})$
  $\boldsymbol{v'} \leftarrow (v'_0, \ldots, v'_{79})$
  for i **from** 0 **to** 287 **do**
    ModelPoly($\mathcal{M}, s_i, Ps[i], \boldsymbol{k'}, \boldsymbol{v'}, \boldsymbol{c}^k, \boldsymbol{c}^v$)
  Return $\boldsymbol{s}$
end procedure

procedure TriviumCore($\mathcal{M}, \boldsymbol{s}, i_1, i_2, i_3, i_4, i_5$)
  $\mathcal{M}.var \leftarrow y_1, y_2, y_3, y_4, y_5, z_1, z_2, a$ as binary variables
  $\mathcal{M}.con \leftarrow s_{i_1} = y_1 \vee z_1$
  $\mathcal{M}.con \leftarrow s_{i_2} = y_2 \vee z_2$
  $\mathcal{M}.con \leftarrow s_{i_3} = y_3 \vee a$
  $\mathcal{M}.con \leftarrow s_{i_4} = y_4 \vee a$
  $\mathcal{M}.con \leftarrow y_5 = s_{i_5} + a + z_1 + z_2$
  $s_{i_j} \leftarrow y_j$ for $j \in \{1, 2, 3, 4, 5\}$
end procedure

procedure ModelPoly($\mathcal{M}, p, f, \boldsymbol{k'}, \boldsymbol{v'}, \boldsymbol{c}^k, \boldsymbol{c}^v$)
  $\mathcal{M}.LinExpr \leftarrow g = 0$
  $\mathcal{M}.var \leftarrow t$ as binary variable
  for each monomial $\boldsymbol{k^{\mu} v^{\nu}}$ **in** f **do**
    for l **from** 0 **to** $n-1$ **do**
      if $\mu_l = 1$ **then**
        $\mathcal{M}.con \leftarrow t = k'_{l,c_l^k}$
        $c_l^k \leftarrow c_l^k + 1$
    for l **from** 0 **to** 79 **do**
      if $\nu_l = 1$ **then**
        $\mathcal{M}.con \leftarrow t = v'_{l,c_l^v}$
        $c_l^v \leftarrow c_l^v + 1$
    $g \leftarrow g + t$
  $\mathcal{M}.con \leftarrow p = g$
end procedure

Algorithm 3: The New MILP Model for Kreyvium

procedure KreyviumModel(Round R, I, Monomial $t = \pi_{\boldsymbol{\omega}}(\boldsymbol{s}) \cdot \pi_{\boldsymbol{\mu}}(\boldsymbol{K}) \cdot \pi_{\boldsymbol{\nu}}(\boldsymbol{V})$)
  $\boldsymbol{Ps}$, $\boldsymbol{g}^{(k)} \leftarrow$ ParamSelector64(I)
  $n \leftarrow \boldsymbol{g}^{(k)}.size()$ ▷ Assign the number of the secret key variables to n
  Prepare an empty MILP Model $\mathcal{M}_{64}$
  $\mathcal{M}_{64}.var \leftarrow \boldsymbol{k} = (k_0, \ldots, k_{n-1})$ as binary variables
  $\mathcal{M}_{64}.var \leftarrow \boldsymbol{v} = (v_0, \ldots, v_{127})$ as binary variables
  $\mathcal{M}_{64}.var \leftarrow \boldsymbol{s} = (s_0, \ldots, s_{287})$ as binary variables
  $\mathcal{M}_{64}.var \leftarrow \boldsymbol{K} = (K_0, \ldots, K_{127})$ as binary variables
  $\mathcal{M}_{64}.var \leftarrow \boldsymbol{V} = (V_0, \ldots, V_{127})$ as binary variables
  $\boldsymbol{s}, \boldsymbol{K}, \boldsymbol{V} \leftarrow$ KreyviumInit64($\mathcal{M}_{64}, \boldsymbol{Ps}, \boldsymbol{s}, \boldsymbol{k}, \boldsymbol{v}, \boldsymbol{K}, \boldsymbol{V}, I, n$)
  for r **from** 65 **to** R **do**
    $\mathcal{M}_{64}.var \leftarrow a, b, c, d, t_1, t_2$ as binary variables
    TriviumCore($\mathcal{M}_{64}, \boldsymbol{s}, 65, 170, 90, 91, 92$)
    TriviumCore($\mathcal{M}_{64}, \boldsymbol{s}, 161, 263, 174, 175, 176$)
    TriviumCore($\mathcal{M}_{64}, \boldsymbol{s}, 242, 68, 285, 286, 287$)
    $\mathcal{M}_{64}.con \leftarrow a \vee b = K_{(r-1) \mod 128}$
    $\mathcal{M}_{64}.con \leftarrow c \vee d = V_{(r-1) \mod 128}$
    $K_{(r-1) \mod 128} \leftarrow a$
    $V_{(r-1) \mod 128} \leftarrow c$
    $\mathcal{M}_{64}.con \leftarrow t_1 = s_{92} + d$
    $\mathcal{M}_{64}.con \leftarrow t_2 = s_{287} + b$
    $\boldsymbol{s} \leftarrow (t_2, s_0, \ldots, s_{91}, t_1, s_{93}, \ldots, s_{286})$
  if $\boldsymbol{\omega} = (0, \ldots, 0)$ **and** $\boldsymbol{\nu} = (0, \ldots, 0)$ **then**
    $\mathcal{M}_{64}.con \leftarrow s_i = 0,\ i \in \{0, \ldots, 287\} \setminus \{65, 92, 161, 176, 242, 287\}$
    $\mathcal{M}_{64}.con \leftarrow K_i = 0,\ i \in \{0, \ldots, 127\}$
    $\mathcal{M}_{64}.con \leftarrow V_i = 0,\ i \in \{0, \ldots, 127\}$
    $\mathcal{M}_{64}.con \leftarrow s_{65} + s_{92} + s_{161} + s_{176} + s_{242} + s_{287} = 1$
  else
    $\mathcal{M}_{64}.con \leftarrow s_i = \omega_i,\ i \in \{0, \ldots, 287\}$
    $\mathcal{M}_{64}.con \leftarrow K_i = \mu_i,\ i \in \{0, \ldots, 127\}$
    $\mathcal{M}_{64}.con \leftarrow V_i = \nu_i,\ i \in \{0, \ldots, 127\}$
  return $\mathcal{M}_{64}, \boldsymbol{g}^{(k)}$
end procedure

procedure KreyviumInit64($\mathcal{M}, \boldsymbol{Ps}, \boldsymbol{s}, \boldsymbol{k}, \boldsymbol{v}, \boldsymbol{K}, \boldsymbol{V}, I, n$)
  $\boldsymbol{n}^k, \boldsymbol{n}^v \leftarrow$ **CountNum**($\boldsymbol{Ps}$) ▷ Count the number of $\boldsymbol{k}$ and $\boldsymbol{v}$ in the polynomials $\boldsymbol{Ps}$
  $n_i^k \leftarrow n_i^k + 1$ for $i \in \{0, \ldots, 127\}$
  $n_i^v \leftarrow n_i^v + 1$ for $i \in I$
  for i **from** 0 **to** $n-1$ **do**
    if $n_i^{(k)} = 0$ **then**
      $\mathcal{M}.con \leftarrow k_i = 0$
    else
      $\mathcal{M}.var \leftarrow k'_i = (k'_{i,1}, k'_{i,2}, \ldots, k'_{i,n_i^{(k)}})$ as binary variables
      $\mathcal{M}.con \leftarrow k_i = k'_{i,1} \vee \cdots \vee k'_{i,n_i^{(k)}}$
      $\mathcal{M}.con \leftarrow K_{127-i} = k'_{i,n_i^{(k)}}$
  for i **from** 0 **to** 127 **do**
    if $n_i^{(v)} = 0$ **then**
      $\mathcal{M}.con \leftarrow v_i = 0$
    else
      $\mathcal{M}.con \leftarrow v_i = 1$
      $\mathcal{M}.var \leftarrow v'_i = (v'_{i,1}, v'_{i,2}, \ldots, v'_{i,n_i^{(v)}})$ as binary variables
      $\mathcal{M}.con \leftarrow v_i = v'_{i,1} \vee \cdots \vee v'_{i,n_i^{(v)}}$
      $\mathcal{M}.con \leftarrow V_{127-i} = v'_{i,n_i^{(k)}}$
  $\boldsymbol{c}^k \leftarrow (1, \ldots, 1) \in \mathbb{Z}^n$
  $\boldsymbol{c}^v \leftarrow (1, \ldots, 1) \in \mathbb{Z}^{128}$
  $\boldsymbol{k'} \leftarrow (k'_0, \ldots, k'_{n-1})$
  $\boldsymbol{v'} \leftarrow (v'_0, \ldots, v'_{127})$
  for i **from** 0 **to** 287 **do**
    ModelPoly($\mathcal{M}, s_i, Ps[i], \boldsymbol{k'}, \boldsymbol{v'}, \boldsymbol{c}^k, \boldsymbol{c}^v$)
  Return $\boldsymbol{s}$, $\boldsymbol{K}$, $\boldsymbol{V}$
end procedure

4 The results of verification experiments on Trivium-like ciphers

4.1 Verification experiments on Kreyvium

For Kreyvium, the target rounds are from 776 to 779, and the sets of cube indices we used come from [6]. Table 1 shows the statistics of the results and provides experimental proof of the effectiveness of the new modeling technique for Kreyvium. Similar to Trivium, the improved MILP model $\mathcal{M}_{196}$ has the fewest feasible solutions and the fastest solving speed among the three different models; the traditional model is the worst performer. In addition, the superpolies recovered by the improved models have the lowest algebraic degree and fewest monomials than the other models.

4.2 Cube indices sets used in verification experiments

Table 1: statistics of the number of the three-subset division trails and the solution time for different MILP models of round-reduced Kreyvium

R	I	# of trails			time (s)			# of monomials in $p_I(\cdot)$			degree of $p_I(\cdot)$		
		$\mathcal{M}_T$	$\mathcal{M}_{64}$	$\mathcal{M}_{192}$	$\mathcal{M}_T$	$\mathcal{M}_{64}$	$\mathcal{M}_{192}$	$\boldsymbol{k}$	$\boldsymbol{x}$	$\boldsymbol{x}^{(2)}$	$\boldsymbol{k}$	$\boldsymbol{x}$	$\boldsymbol{x}^{(2)}$
776	I_{64}	3	1	1	52	57	40	3	1	1	2	1	1
	I_{65}	435	9	3	62	58	38	3	1	1	2	1	1
	I_{66}	3	1	1	68	59	37	3	1	1	2	1	1
	I_{67}	165	3	3	70	62	42	3	1	1	2	1	1
	I_{68}	3	1	1	60	54	41	3	1	1	2	1	1
	I_{69}	393	23	7	58	58	42	3	1	1	2	1	1
	I_{70}	3	1	1	57	59	39	3	1	1	2	1	1
	I_{71}	3	1	1	59	52	38	3	1	1	2	1	1
777	I_{64}	0	0	0	56	53	36	0	0	0	$-\infty$	$-\infty$	$-\infty$
	I_{65}	0	0	0	55	56	37	0	0	0	$-\infty$	$-\infty$	$-\infty$
	I_{66}	97686	150	25	834	59	43	108	4	1	8	5	4
	I_{67}	4650	91	29	526	62	52	12	3	1	3	2	2
	I_{68}	3132	44	2	76	67	37	0	0	0	$-\infty$	$-\infty$	$-\infty$
	I_{69}	4666	35	12	381	72	42	1256	23	6	9	8	5
	I_{70}	2484	42	4	65	63	45	792	6	4	10	6	5
	I_{71}	0	0	0	71	53	37	0	0	0	$-\infty$	$-\infty$	$-\infty$
778	I_{64}	58590	164	10	479	55	38	5850	60	2	13	10	7
	I_{65}	13893	264	15	724	62	39	3279	174	9	11	9	6
	I_{66}	u	85259	12274	3600	1577	410	1219899	10573	1906	17	12	9
	I_{67}	u	1563	95	3600	80	50	144568	993	65	15	12	8
	I_{68}	u	7571	703	3600	500	55	10770	457	67	10	8	5
	I_{69}	u	9593	1746	3600	1647	70	110682	1501	280	12	8	6
	I_{70}	u	6808	701	3600	416	49	33720	920	91	10	8	5
	I_{71}	u	2579	411	3600	84	50	86242	773	123	15	11	8
779	I_{64}	81	3	1	68	55	37	81	3	1	8	5	4
	I_{65}	5079	131	37	564	67	58	405	27	9	8	5	4
	I_{67}	u	2306	845	3600	434	56	18744	728	215	12	9	6
	I_{68}	u	93445	15506	3600	1396	699	384612	8273	2208	14	9	7
	I_{69}	u	19591	3988	3600	1141	91	75594	2529	696	12	9	6
	I_{70}	u	52748	6568	3600	2235	421	231804	4590	938	14	10	7
	I_{71}	u	593	60	3600	75	46	85563	265	28	15	10	8

Table 2: cube indices used in practical verification

I_4	2, 5, 6, 9, 13, 16, 19, 21, 23, 25, 27, 29, 30, 32, 34, 36, 38, 40, 42, 44, 45, 48, 53, 57, 59, 61, 65, 68, 73, 75, 78
I_5	1, 3, 6, 8, 11, 14, 15, 18, 22, 25, 27, 29, 34, 37, 40, 42, 46, 48, 50, 52, 55, 57, 59, 61, 66, 68, 69, 71, 74, 79
I_6	2, 4, 6, 8, 10, 13, 15, 19, 22, 24, 28, 29, 32, 34, 37, 38, 40, 41, 44, 47, 49, 51, 53, 55, 57, 59, 62, 70, 73, 78
I_7	2, 3, 4, 6, 8, 9, 10, 14, 16, 19, 21, 22, 23, 25, 28, 29, 30, 34, 36, 37, 39, 41, 46, 47, 48, 51, 56, 59, 64, 68, 71, 74, 79
I_8	2, 5, 6, 9, 11, 13, 16, 19, 21, 23, 24, 25, 27, 29, 30, 32, 34, 36, 38, 40, 42, 44, 45, 48, 51, 53, 55, 57, 59, 68, 73
I_9	2, 4, 6, 8, 10, 12, 14, 16, 19, 21, 22, 23, 25, 29, 30, 32, 34, 39, 41, 46, 47, 48, 49, 51, 56, 59, 64, 67, 68, 71, 79
I_{10}	1, 3, 6, 11, 14, 16, 20, 22, 24, 27, 30, 32, 35, 37, 39, 42, 44, 46, 47, 48, 49, 50, 53, 55, 59, 64, 68, 70, 72, 78
I_{11}	1, 2, 3, 6, 8, 11, 14, 17, 20, 22, 24, 27, 32, 34, 37, 39, 42, 44, 46, 47, 48, 50, 53, 55, 57, 59, 62, 64, 68, 70, 72
I_{12}	2, 3, 6, 8, 11, 13, 14, 16, 20, 22, 24, 27, 30, 32, 34, 35, 37, 39, 42, 44, 46, 48, 50, 53, 55, 57, 59, 62, 68, 72, 78
I_{13}	2, 4, 6, 8, 10, 12, 14, 16, 21, 23, 25, 30, 32, 34, 36, 37, 39, 41, 46, 47, 48, 49, 51, 56, 59, 64, 67, 68, 71, 79
I_{14}	2, 5, 6, 10, 13, 16, 21, 23, 25, 27, 29, 34, 36, 38, 40, 42, 44, 45, 48, 51, 53, 55, 59, 61, 65, 68, 70, 73, 75, 78
I_{15}	2, 4, 6, 8, 10, 12, 13, 15, 19, 20, 22, 24, 26, 31, 34, 37, 38, 42, 44, 47, 49, 53, 55, 57, 59, 68, 70, 73, 76, 78
I_{16}	4, 6, 8, 10, 13, 15, 19, 20, 24, 26, 28, 31, 34, 37, 38, 40, 41, 42, 44, 47, 49, 51, 53, 55, 57, 59, 68, 70, 73, 76, 78
I_{17}	1, 4, 6, 8, 10, 13, 15, 19, 20, 22, 24, 26, 28, 29, 32, 34, 38, 40, 41, 42, 49, 51, 53, 55, 57, 59, 62, 68, 70, 76, 78
I_{18}	2, 5, 9, 10, 13, 16, 21, 23, 25, 27, 29, 30, 32, 34, 36, 38, 40, 42, 44, 45, 48, 53, 55, 57, 59, 63, 65, 68, 75, 78
I_{19}	2, 4, 6, 8, 10, 13, 15, 19, 20, 24, 26, 31, 34, 37, 38, 40, 42, 44, 47, 49, 51, 53, 55, 57, 59, 68, 70, 73, 76, 78
I_{20}	2, 5, 6, 9, 13, 16, 23, 25, 27, 29, 30, 34, 36, 38, 40, 42, 44, 45, 48, 51, 53, 55, 57, 59, 61, 63, 65, 68, 70, 78
I_{21}	1, 4, 6, 8, 10, 12, 13, 15, 19, 20, 22, 24, 26, 28, 31, 34, 38, 40, 41, 42, 44, 47, 49, 55, 57, 59, 68, 70, 73, 76, 78
I_{22}	2, 4, 6, 8, 10, 13, 15, 20, 22, 24, 26, 28, 31, 34, 37, 38, 40, 42, 44, 47, 49, 51, 53, 55, 57, 59, 62, 68, 70, 73, 78
I_{23}	2, 4, 6, 8, 10, 13, 15, 19, 20, 22, 24, 26, 28, 29, 31, 32, 34, 37, 38, 40, 42, 44, 47, 51, 53, 57, 59, 62, 70, 73, 76, 78
I_{24}	1, 4, 6, 8, 10, 13, 15, 19, 20, 22, 24, 26, 28, 29, 31, 34, 38, 40, 42, 44, 49, 51, 53, 55, 57, 59, 68, 70, 76, 78
I_{25}	2, 4, 6, 8, 15, 19, 20, 22, 24, 26, 29, 31, 32, 34, 37, 38, 40, 42, 44, 47, 49, 51, 53, 55, 57, 59, 68, 70, 76, 78
I_{26}	2, 4, 6, 8, 13, 15, 19, 20, 22, 24, 26, 28, 31, 32, 34, 40, 41, 42, 47, 49, 51, 53, 55, 57, 59, 68, 70, 73, 76, 78
I_{27}	2, 4, 6, 8, 10, 13, 15, 19, 20, 22, 24, 26, 28, 29, 31, 32, 34, 40, 42, 44, 47, 49, 53, 55, 57, 59, 68, 70, 76, 78
I_{28}	2, 4, 6, 8, 10, 12, 15, 19, 20, 22, 24, 26, 28, 29, 32, 34, 37, 40, 42, 44, 47, 51, 53, 55, 57, 59, 62, 68, 70, 78
I_{29}	1, 2, 6, 8, 11, 13, 14, 17, 20, 22, 24, 27, 32, 34, 37, 39, 42, 44, 46, 47, 48, 50, 53, 55, 57, 59, 62, 64, 68, 70, 78
I_{30}	2, 4, 6, 8, 10, 12, 13, 15, 19, 24, 28, 29, 32, 34, 37, 40, 41, 44, 47, 49, 51, 53, 55, 57, 59, 62, 65, 70, 72, 73, 74, 76, 78
I_{31}	2, 4, 6, 8, 10, 12, 13, 15, 19, 24, 28, 29, 32, 34, 37, 38, 40, 41, 44, 47, 49, 51, 53, 55, 57, 59, 68, 70, 72, 73, 74, 76, 78
I_{32}	0, 2, 4, 5, 6, 7, 9, 11, 13, 14, 15, 18, 20, 22, 24, 26, 32, 35, 37, 39, 42, 44, 46, 48, 53, 55, 57, 61, 68, 70, 72, 79
I_{33}	0, 2, 4, 5, 6, 7, 9, 11, 13, 14, 15, 18, 20, 22, 24, 26, 32, 35, 39, 42, 44, 46, 48, 52, 55, 57, 62, 68, 70, 74, 76, 79
I_{34}	0, 2, 4, 5, 7, 9, 11, 13, 14, 15, 18, 20, 24, 26, 30, 32, 35, 37, 39, 40, 42, 44, 46, 48, 52, 53, 55, 62, 68, 70, 74, 79
I_{35}	0, 4, 5, 6, 7, 9, 11, 13, 14, 18, 20, 22, 24, 26, 30, 35, 37, 39, 40, 44, 46, 48, 52, 55, 57, 62, 68, 70, 72, 74, 76, 79
I_{36}	0, 2, 4, 5, 6, 9, 11, 13, 14, 15, 17, 18, 19, 20, 22, 24, 26, 30, 32, 35, 39, 40, 44, 48, 53, 55, 57, 61, 62, 70, 74, 76, 79
I_{37}	0, 5, 6, 7, 9, 11, 13, 17, 19, 22, 24, 26, 30, 32, 35, 37, 39, 42, 44, 46, 48, 52, 53, 55, 57, 61, 62, 68, 72, 74, 76, 79
I_{38}	0, 2, 4, 5, 6, 7, 9, 11, 13, 15, 17, 18, 19, 22, 24, 26, 30, 32, 37, 39, 42, 44, 46, 52, 53, 57, 61, 62, 68, 74, 76, 79
I_{39}	0, 4, 5, 7, 9, 11, 13, 14, 15, 17, 18, 19, 20, 22, 24, 26, 30, 32, 35, 37, 39, 40, 44, 48, 53, 55, 61, 68, 72, 74, 76, 79
I_{40}	0, 4, 5, 6, 7, 9, 11, 13, 15, 18, 20, 22, 24, 26, 30, 32, 35, 37, 39, 40, 42, 44, 46, 48, 55, 57, 62, 68, 70, 72, 76, 79
I_{41}	0, 4, 5, 6, 7, 9, 11, 13, 14, 15, 17, 19, 22, 24, 26, 32, 35, 37, 39, 40, 42, 46, 48, 52, 55, 57, 62, 68, 70, 74, 76, 79
I_{42}	0, 2, 4, 6, 8, 11, 13, 16, 19, 21, 23, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 49, 50, 53, 56, 62, 64, 69, 72, 74, 75, 77, 79
I_{43}	0, 2, 4, 6, 8, 11, 13, 16, 19, 21, 23, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 50, 53, 56, 58, 62, 64, 69, 72, 74, 75, 77, 79
I_{44}	0, 2, 4, 6, 8, 11, 13, 16, 19, 21, 23, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 49, 50, 53, 56, 58, 59, 62, 69, 71, 74, 75, 79
I_{45}	0, 2, 4, 6, 8, 11, 13, 16, 19, 21, 23, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 49, 50, 53, 56, 59, 62, 64, 66, 69, 72, 74, 75, 77, 79
I_{46}	0, 2, 4, 6, 8, 11, 13, 16, 19, 21, 23, 26, 28, 32, 34, 36, 38, 40, 42, 44, 46, 50, 53, 56, 58, 59, 62, 64, 66, 69, 71, 72, 74, 75, 77, 79
I_{47}	0, 2, 4, 6, 8, 11, 13, 16, 19, 21, 23, 26, 28, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 53, 56, 58, 59, 62, 66, 69, 71, 72, 74, 75, 77, 79
I_{48}	0, 2, 4, 5, 6, 7, 9, 11, 13, 14, 15, 18, 20, 22, 24, 26, 32, 35, 37, 39, 42, 44, 48, 52, 53, 55, 57, 61, 62, 68, 70, 74, 79
I_{49}	2, 5, 7, 9, 13, 17, 19, 22, 24, 28, 30, 37, 39, 41, 43, 45, 52, 54, 58, 64, 69, 71, 73, 77, 81, 83, 87, 92, 97, 103, 106, 109, 117, 121
I_{50}	0, 2, 5, 7, 9, 13, 19, 22, 24, 28, 30, 37, 39, 41, 43, 45, 52, 54, 58, 66, 69, 71, 73, 77, 81, 83, 87, 92, 97, 103, 106, 109, 117, 121, 127
I_{51}	0, 2, 5, 7, 9, 13, 17, 19, 22, 24, 28, 30, 37, 39, 41, 43, 45, 49, 52, 54, 64, 66, 69, 71, 73, 77, 81, 83, 97, 103, 106, 117, 121, 127
I_{52}	2, 5, 7, 9, 13, 17, 19, 22, 24, 28, 30, 37, 39, 41, 43, 45, 49, 52, 64, 66, 69, 71, 73, 77, 81, 83, 87, 92, 97, 103, 106, 109, 117, 121
I_{53}	0, 2, 5, 7, 9, 13, 17, 19, 22, 24, 28, 30, 37, 39, 41, 43, 45, 49, 52, 54, 64, 66, 69, 71, 73, 77, 81, 83, 87, 92, 97, 103, 106, 117, 127
I_{54}	0, 2, 5, 7, 13, 17, 19, 22, 24, 28, 30, 37, 41, 43, 45, 49, 52, 54, 58, 64, 66, 71, 73, 77, 81, 83, 87, 92, 97, 103, 106, 109, 117, 121, 127
I_{55}	0, 2, 5, 7, 9, 13, 17, 19, 22, 24, 28, 30, 37, 39, 41, 43, 45, 52, 54, 64, 66, 69, 71, 73, 77, 81, 83, 87, 92, 97, 103, 106, 109, 117, 127
I_{56}	2, 5, 7, 9, 13, 17, 19, 22, 24, 28, 30, 37, 39, 41, 43, 45, 52, 54, 58, 64, 66, 69, 71, 73, 77, 81, 83, 87, 97, 103, 109, 117, 121, 127

5 Key-recovery attack against 849-round Trivium

Although we have recovered the superpoly for 849-round Trivium, it is very complex and contains all 80 key variables, making it difficult to recover the key information using this polynomial. To solve this problem, He et al. proposed a key recovery strategy in [5], which can be seen as embedding the key testing procedure into a Möbius transformation with adjusted computational order, and based on this strategy, proposed Algorithm 4. We first introduce the required notation before describing this algorithm.

Let $f(\boldsymbol{x}) = \bigoplus_{(c_0,\ldots,c_{n-1})\in\mathbb{F}_2^n} g_0(c_0,\ldots,c_{n-1}) \cdot \prod_{i=0}^{n-1} x_i^{c_i} \in \mathbb{F}_2[\boldsymbol{x}]$, where $\boldsymbol{x} = (x_0,\ldots,x_{n-1}) \in \mathbb{F}_2^n$. According to the process of the Möbius transformation from ANF to the truth table, f can be represented as follows,

$$\begin{aligned} f &= \bigoplus_{(c_0,\ldots,c_{n-1})\in\mathbb{F}_2^n} g_0(c_0,\ldots,c_{n-1}) \cdot \prod_{i=0}^{n-1} x_i^{c_i} \\ &= \bigoplus_{(c_1,\ldots,c_{n-1})\in\mathbb{F}_2^{n-1}} g_1(x_0,c_1,\ldots,c_{n-1}) \cdot \prod_{i=1}^{n-1} x_i^{c_i} \\ &\vdots \\ &= \bigoplus_{(c_t,\ldots,c_{n-1})\in\mathbb{F}_2^{n-t}} g_t(x_0,\ldots,x_{t-1},c_t,\ldots,c_{n-1}) \cdot \prod_{i=t}^{n-1} x_i^{c_i}, \end{aligned}$$

where $t \in \{1,\ldots,n\}$ and $g_t(x_0,\ldots,x_{t-1},c_t,\ldots,c_{n-1})$ is the Boolean function on $x_0,\ldots,x_{t-1}$, which satisfies the following equation,

$$\begin{aligned} g_t(x_0,\ldots,x_{t-1},c_t,\ldots,c_{n-1}) = {} & g_{t-1}(x_0,\ldots,x_{t-2},0,c_t,\ldots,c_{n-1}) \\ & \oplus g_{t-1}(x_0,\ldots,x_{t-2},1,c_t,\ldots,c_{n-1}) \cdot x_{t-1}. \end{aligned}$$

For simplicity, we use $g_t(e)(0 \le t \le n)$ to represent $g_t(c_0,\ldots,c_{n-1})$, where $e = c_0 + c_1 2 + \cdots + c_{n-1}2^{n-1}$ and $(c_0,\ldots,c_{n-1})$ is called the binary representation of e.

Based on the above notation, we show Algorithm 4 as follows, which can be used to recover the key from the equation $f(x_0,\ldots,x_{n-1}) = a$. Though a more accurate estimation of the average complexity can be given, the time cost of Algorithm 4 are roughly estimated as one Möbius transformation together with required encryption calls, that is, $q \cdot 2^n$ encryption calls $+n \cdot 2^{n-1}$ XORs, where q denotes the probability that $f(x_0,\ldots,x_{n-1}) = a$. The cost of the comparison is ignored. The memory complexity in the worst case is 2^n bits. Compared with the traditional key recovery method of first constructing a large truth table and then performing queries, this method naturally saves the query cost.

For 849-round Trivium, the superpoly we recovered is represented as the polynomial of the new secret key variables $\boldsymbol{x} \in \mathbb{F}_2^\alpha$, denoted by $p(x_0,\ldots,x_{\alpha-1})$, which contains about $2^{22.78}$ terms and whose degree is 15. A natural idea is to

Algorithm 4: Recover the key from the equation

procedure RecoverKey(The ANF of f, the value of f a)
  for t **from** 1 **to** n **do**
    Precompute the ANF of $g_{t-1}(x_0, \ldots, x_{t-2}, 1, 0, \ldots, 0)$ from the ANF of f
  if $g_0(0, \ldots, 0) = a$ **then**
    return $(0, \ldots, 0)$
  for t **from** 1 **to** n **do**
    Perform Möbius transformation on the ANF of $g_{t-1}(x_0, \ldots, x_{t-2}, 1, 0, \ldots, 0)$ to obtain the truth table of $g_{t-1}(x_0, \ldots, x_{t-2}, 1, 0, \ldots, 0)$
    for k **from** 0 **to** $2^{t-1} - 1$ **do**
      $g_t(2^{t-1} + k) = g_{t-1}(2^{t-1} + k) \oplus g_{t-1}(k)$
      if $g_t(2^{t-1} + k) = a$ **then**
        Let $(c_0, \ldots, c_{n-1})$ be the binary representation of $2^{t-1} + k$
        Check if $(c_0, \ldots, c_{n-1})$ is the correct key by calling the encryption oracle once
        if $g_t(2^{t-1} + k) = a$ **then**
          return $(c_0, \ldots, c_{n-1})$
        $g_t(k) = g_{t-1}(k)$
  return no solution found
end procedure

first represent $p(x_0, \ldots, x_{\alpha-1})$ as the polynomial of $\boldsymbol{k}$, and then to treat it as f in Algorithm 4. However, Möbius transformation may also incur memory access costs. In the worse case, Algorithm 4 requires 2^{80}-bits memory, and the memory access cost of such a big table is unbearable. As shown below, we adopt a similar strategy to [5] to address this difficulty.

1. According to the new modeling technique, we have the algebraic relations between $\boldsymbol{x} = (x_0, \ldots, x_{\alpha-1})$ and $\boldsymbol{k} = (k_0, \ldots, k_{79})$, i.e.,

$$x_i = k_i, \text{ for } i \in \{0, \ldots, 79\},$$
$$x_i = h_i(k_0, \ldots, k_{79}) \text{ for } i \in \{80, \ldots, \alpha - 1\}.$$

   Firstly, we guess the values of $(k_{40}, \ldots, k_{79})$. Let $(a_{40}, \ldots, a_{79})$ denote the values of $(k_{40}, \ldots, k_{79})$. Hence, the superpoly $p(x_0, \ldots, x_{\alpha-1}) = a$ can be reduced to

$$p'(x_0, \ldots, x_{39}, x_{80}, \ldots, x_{\alpha-1}) = p(x_0, \ldots, x_{39}, a_{40}, \ldots, a_{79}, x_{80}, \ldots, x_{\alpha-1}) = a$$

   . Then, according to $x_i = h_i(k_0, \ldots, k_{79})$ for $i \in \{80, \ldots, \alpha - 1\}$, we can obtain $p'(k_0, \ldots, k_{39}) = p(k_0, \ldots, k_{39}, a_{40}, \ldots, a_{79}) = a$. Note that through roughly estimating, the total number of monomials involved in $p'(k_0, \ldots, k_{39})$ does not exceed $2^{27.29}$ for each $(a_{40}, \ldots, a_{79}) \in \mathbb{F}_2^{40}$.
2. For each guess, treat $p'(k_0, \ldots, k_{39})$ as f and apply Algorithm 4 to it. Once the algorithm returns the correct values of $(k_0, \ldots, k_{39})$, denoted by $(a_0, \ldots, a_{39})$, the correct key is found as $(a_0, \ldots, a_{79})$.

Assuming reducing $p(x_0, \ldots, x_{\alpha-1})$ to $p'(x_0, \ldots, x_{39}, x_{80}, \ldots, x_{\alpha-1})$ and obtaining the equation $p'(k_0, \ldots, k_{39}) = p(k_0, \ldots, k_{39}, a_{40}, \ldots, a_{79})$ from $p'(x_0, \ldots, x_{39}, x_{80}, \ldots, x_{\alpha-1})$ for all guesses require $2^{40} \times (2^{22.78} + 2^{27.29}) \approx 2^{27.35}$ XORs, the final time complexity is approximately 2^{79} Trivium calls and 40×2^{79} XORs. Assuming one 849-round Trivium call is equivalent to $849 \times 9 = 7641$ XORs, finally, the key recovery strategy requires slightly more than 2^{79} 849-round Trivium calls, but only about 2^{40}-bits memory.

References

1. Cannière, C.D., Preneel, B.: Trivium. In: Robshaw, M.J.B., Billet, O. (eds.) New Stream Cipher Designs - The eSTREAM Finalists, Lecture Notes in Computer Science, vol. 4986, pp. 244–266. Springer, Berlin, Heidelberg (2008), https://doi.org/10.1007/978-3-540-68351-3_18
2. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. J. Cryptol. **31**(3), 885–916 (2018). https://doi.org/10.1007/s00145-017-9273-9, https://doi.org/10.1007/s00145-017-9273-9
3. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset - improved cube attacks against Trivium and Grain-128AEAD. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12105, pp. 466–495. Springer, Cham (2020), https://doi.org/10.1007/978-3-030-45721-1_17
4. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset. J. Cryptol. **34**(3), 22 (2021). https://doi.org/10.1007/s00145-021-09383-2, https://doi.org/10.1007/s00145-021-09383-2
5. He, J., Hu, K., Preneel, B., Wang, M.: Stretching cube attacks: Improved methods to recover massive superpolies. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 13794, pp. 537–566. Springer, Cham (2022), https://doi.org/10.1007/978-3-031-22972-5_19
6. Ye, C., Tian, T.: A new framework for finding nonlinear superpolies in cube attacks against Trivium-like ciphers. In: Susilo, W., Yang, G. (eds.) Information Security and Privacy - 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10946, pp. 172–187. Springer, Cham (2018), https://doi.org/10.1007/978-3-319-93638-3_11