Formal Specification and Verification
of
Fully Asynchronous Implementations
of the
Data Encryption Standard

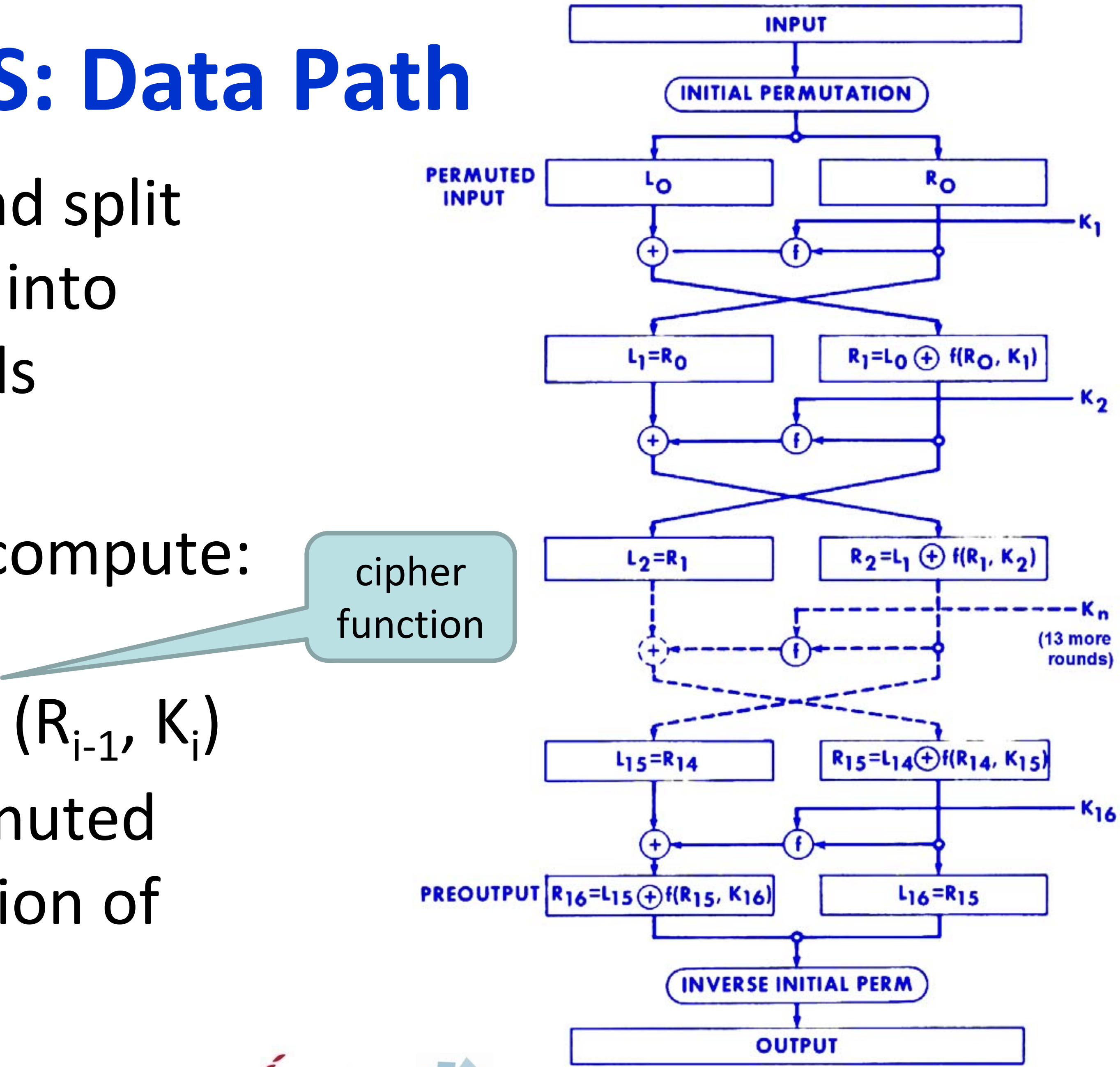
Wendelin Serwe

Data Encryption Standard (DES)

- Symmetric-key block cipher
  - Input: 64-bit data block, 64-bit key, cipher/decipher
  - Output: (de)ciphered 64-bit data block
- FIPS standard 46 for almost 30 years
- Main weakness: only 56 *useful* key bits
- TDEA (or Triple DES)
  - approved block cipher (at least until 2030)
  - recommended for payment systems EMV
  - three applications of the DES with three different keys: cipher, decipher, cipher
- Specified as data-flow diagram

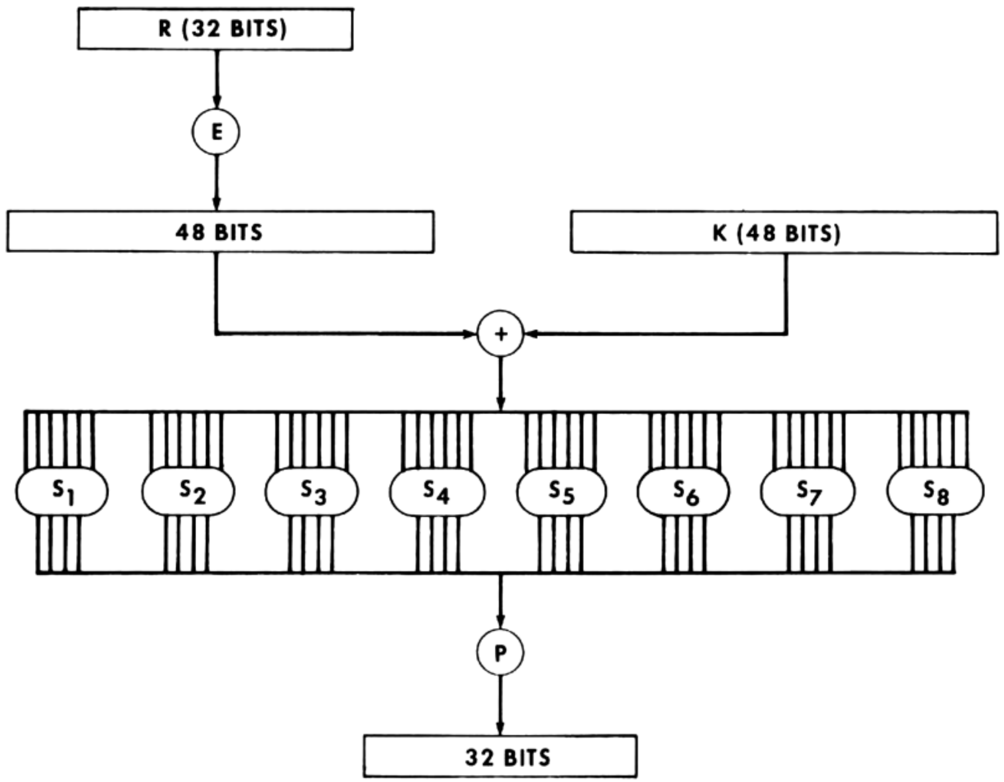
DES: Data Path

- Permute and split 64-bit data into 32-bit words L_0 and R_0
- iteratively compute:
  $L_i := R_{i-1}$
  $R_i := L_{i-1} \oplus f\,(R_{i-1}, K_i)$
- return permuted concatenation of R_{16} and L_{15}



DES: Cipher Function

- Expand R_{i-1} to 48-bit word E (R_{i-1})
- Split E $(R_{i-1}) \oplus K_i$ into eight 6-bit words $X_1 \ldots X_8$
- Compute eight 4-bit words $Y_j := S_j (X_j)$ (using the S-boxes S_j)
- Return permutation of the concatenation $Y_1 \ldots Y_8$

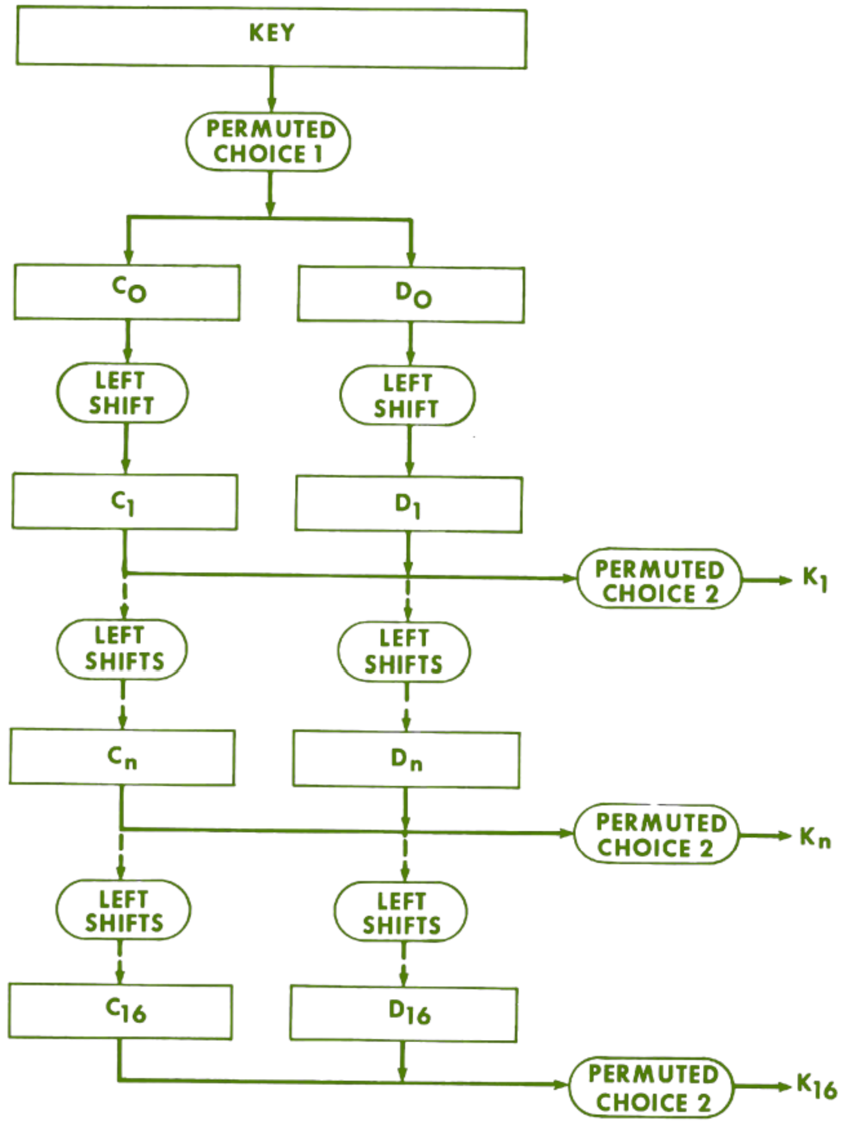


DES: Key Path

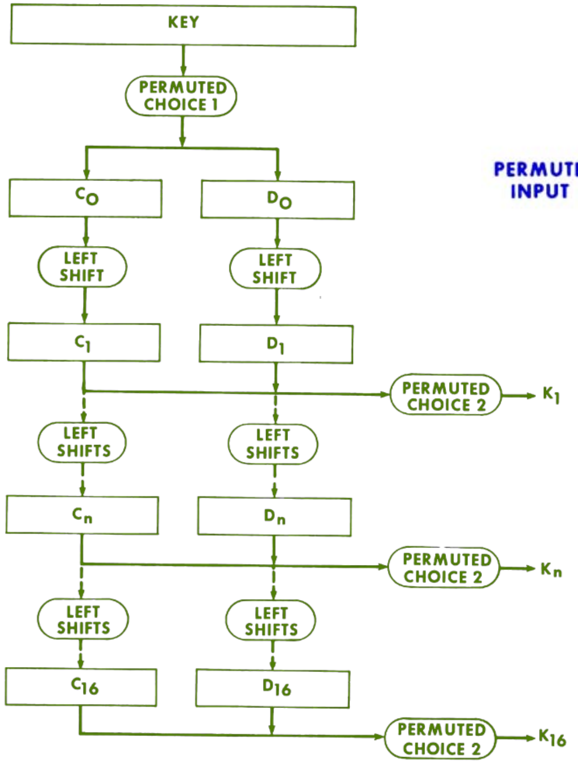
- select and split 64-bit key into 28-bit words C_0 and D_0
- iteratively compute:
  C_i := LeftShift (i, C_{i-1})
  D_i := LeftShift (i, D_{i-1})

shift left by 1 or 2 bits (depending on i)

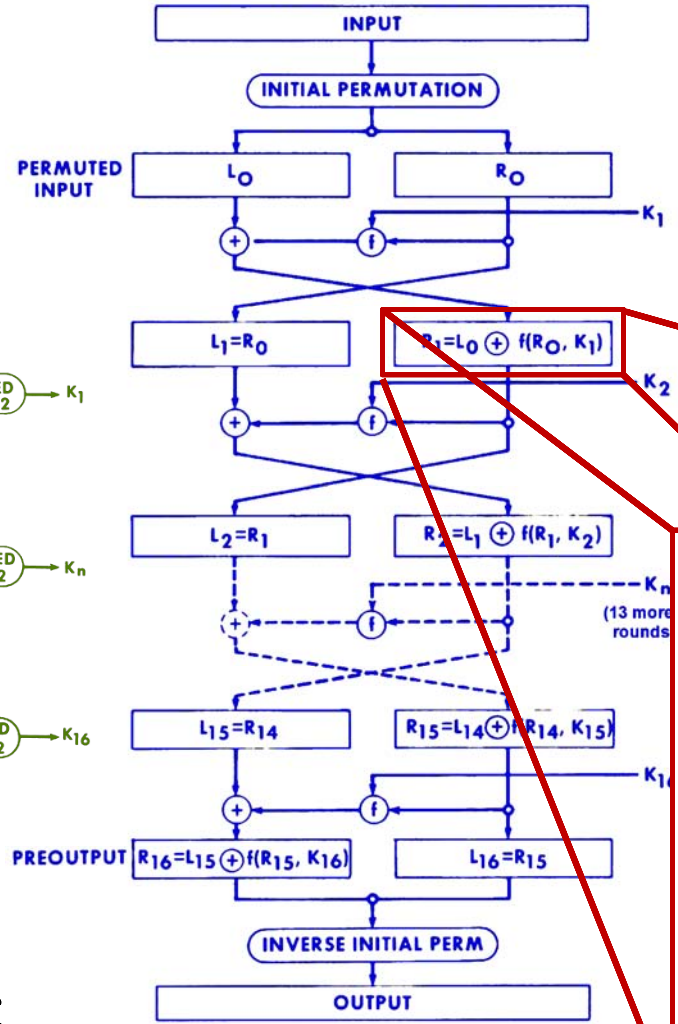
K_i := selection of 48 bits from C_i and D_i



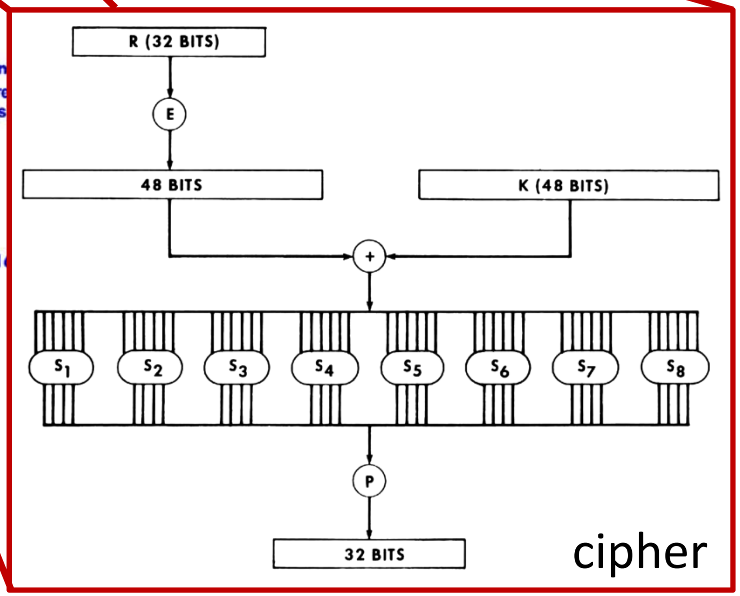
Data-flow Diagram of the DES



key path



data path



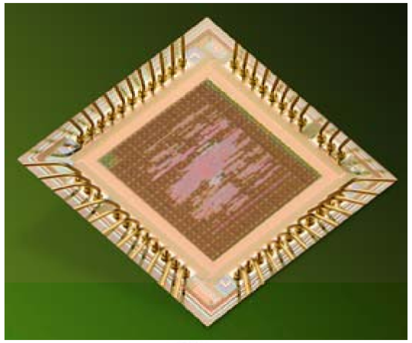
cipher

- highly parallel (S-boxes S_i)
- asynchronous
- nondeterministic execution

- 16 iterations
- deterministic blocks

Related Work

- Asynchronous circuit described in CHP
  - tolerance of low and variable power supply
  - low power consumption
  - resistance to security attacks (side-channel, fault-injection)
- 2003: Translation of simplified CHP to IF
  - Generation of the state space (LTS)
  - Model and equivalence checking (CADP)
- 2008: DES4 chip released



History of our Models

- 2003: LOTOS model
  - derived from the DES standard
  - fully asynchronous
  - compositional LTS generation and verification
  - rapid prototyping
- 2004: development of the CHP2LOTOS compiler
  - application to the CHP model of the DES
  - improved verification performance compared to IF
- 2015: LNT model
  - rewrite of the LOTOS model
  - additional properties

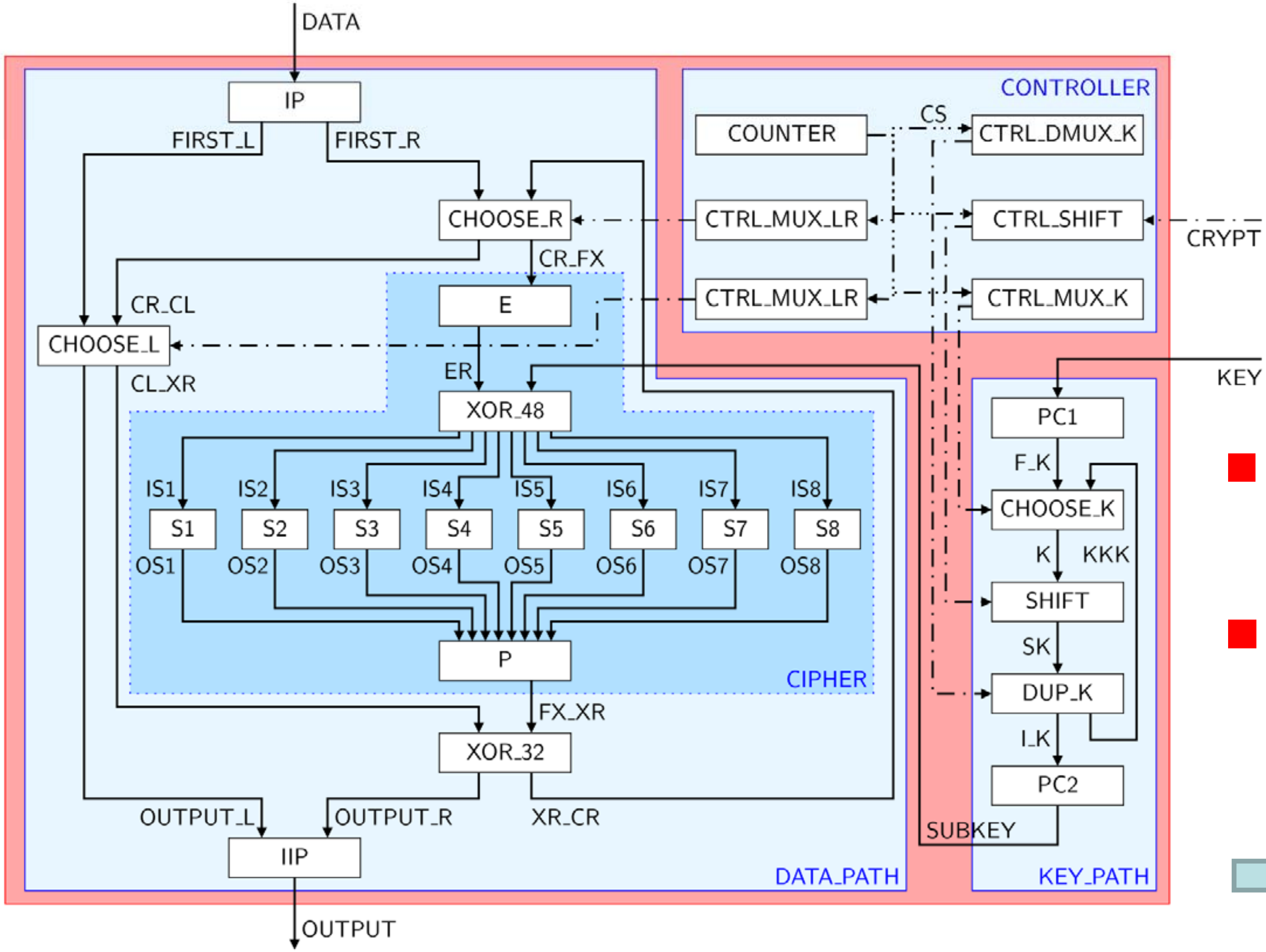
CADP (http://cadp.inria.fr)

- Construction and Analysis of Distributed Processes
- *Modular* toolbox with *several*
  - Formal specification languages:
    LOTOS, LNT, FSP, π-calculus, …
  - Verification paradigms:
    model checking, equivalence checking, visual checking, …
  - Analysis techniques:
    reachability, on-the-fly, compositional, distributed, static analysis, rapid prototyping, test generation, performance evaluation, …
- Continuous development for more than 25 years
- More than 150 case-studies and 70 3rd party tools

LOTOS and LNT

- **LOTOS**
  - international standard ISO:8807 (1988)
  - combination of CCS, CSP and abstract data types
  - powerful, but steep learning curve
- **LNT** [Champelovier-Clerc-Garavel-et-al-10]
  - simplification of international standard E-LOTOS
  - *symmetrical* sequential composition
  - process calculus with imperative syntax
  - development partially funded by industry
  - currently implemented by translation into LOTOS

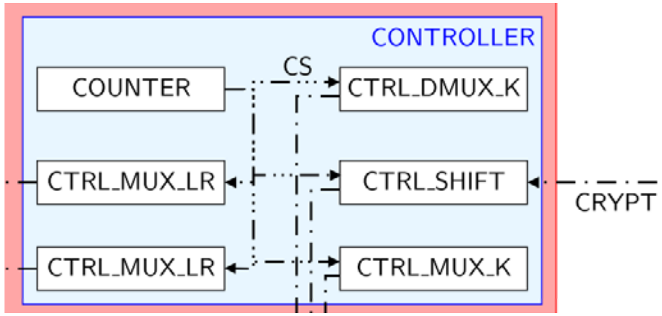
Architecture of the Formal Models



- correspondence: block -> process
- share processes between iterations

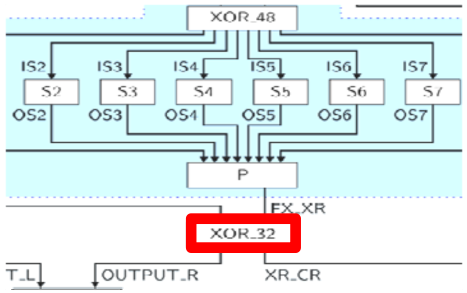
⇨ add control processes & multiplexers

Six Control Processes



- Controller for Shift-Register
  - amount (1 or 2 bits) and
  - shift direction (left or right)
- Four controllers for multiplexers
  - choice of input: initial value or previous iteration
  - output: next iteration or final result
- Counter of the iterations: 0 to 16
  - one value more than the number of iterations
  - multiplexers before and after each iteration

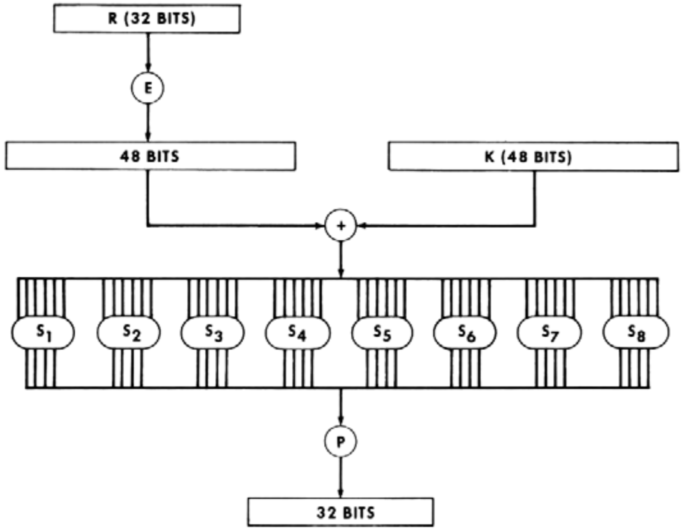
Example: XOR_32




```
process XOR_32 [A, B, R: C32] is
  var A32, B32: BIT32 in
    loop
      par
        A (?A32)
      ||
        B (?B32)
      end par;
      R (XOR (A32, B32))
    end loop
  end var
end process -- LNT
```

```
process XOR_32 [A, B, R] : noexit :=
  (
    A ? A32 : BIT32;
      exit (A32, any BIT32)
    |||
    B ? B32 : BIT32;
      exit (any BIT32, B32)
  )
  >> accept A32, B32 : BIT32 in
    R ! XOR (A32, B32);
      XOR_32 [A, B, R]
endproc                    (* LOTOS *)
```

Example: CIPHER




```
process CIPHER [K: C48, R, PX: C32] is
  hide ER: C48,
        IS1, IS2, IS3, IS4, IS5, IS6, IS7, IS8: C6,
        SO1, SO2, SO3, SO4, SO5, SO6, SO7, SO8: C4 in
    par
      ER ->
        E [R, ER]
    || ER, IS1, IS2, IS3, IS4, IS5, IS6, IS7, IS8 ->
        XOR_48 [ER, K, IS1, IS2, IS3, IS4, IS5, IS6, IS7, IS8]
    || IS1, IS2, IS3, IS4, IS5, IS6, IS7, IS8, SO1, SO2, SO3, SO4, SO5, SO6, SO7, SO8 ->
        par
            S1 [IS1, SO1]    || S2 [IS2, SO2]   || S3 [IS3, SO3]   || S4 [IS4, SO4]
         || S5 [IS5, SO5]    || S6 [IS6, SO6]   || S7 [IS7, SO7]   || S8 [IS8, SO8]
        end par
    || SO1, SO2, SO3, SO4, SO5, SO6, SO7, SO8 ->
        P [SO1, SO2, SO3, SO4, SO5, SO6, SO7, SO8, PX]
    end par
  end hide
end process -- LNT
```

Comparison of LNT and LOTOS Models

	LOTOS	LNT	generated LOTOS
types & functions	1172	575	2514
channels	0	50	58
processes	671	668	772
total	*1843*	*1293*	*3344*

- LNT shorter
- LNT closer to the standard
- generated LOTOS much larger, due to
  - automatically generated data types (S-boxes matrices)
  - automatically generated auxiliary functions

S-Boxes in the Standard

- Compute $S_k(X)$, with $X = B_1B_2B_3B_4B_5B_6$
  - row $i = B_1B_6$, column $j = B_2B_3B_4B_5$
  - return binary representation of $S_k[i, j]$
- Example: $S_1(011011)$
  - $i = 01 = 1$, $j = 1101 = 13$, $S_1[1, 13] = 5$
  - $S_1(011011) = 0101$

S₁

														j = 13		
	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
i = 1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-Boxes in LNT

```
type ROW is array [0..15] of NAT end type
type S_BOX_ARRAY is array [0..3] of ROW end type
function GET_ROW (X: BIT6) : NAT is
  return BIT2_TO_NAT (1AND6 (X))
end function
function GET_COLUMN (X: BIT6) : NAT is
  return BIT4_TO_NAT (2TO5 (X))
end function
function S1 : S_BOX_ARRAY is
  return S_BOX_ARRAY
   (ROW (14,  4, 13,  1,   2, 15, 11,  8,    3, 10,    6, 12,  5,   9,  0,   7),
    ROW (  0, 15,  7,  4, 14,   2, 13,   1, 10,    6, 12, 11,  9,   5,  3,   8),
    ROW (  4,   1, 14, 8, 13,   6,   2, 11, 15, 12,    9,   7,  3, 10,  5,   0),
    ROW (15, 12,  8,  2,   4,   9,   1,   7,   5,  11,   3, 14, 10,  0,  6, 13))
end function
process S1 [INPUT: C6, OUTPUT: C4] is
  loop var I6: BIT6 in
      INPUT (?I6);
      OUTPUT (NAT_TO_BIT4 (S1[GET_ROW (I6)][GET_COLUMN (I6)]))
  end var end loop
end process
```

S-Boxes in LOTOS

```
type S_BOX_FUNCTIONS is BIT4, BIT6
  opns S1 : BIT6 -> BIT4
  eqns
    ofsort BIT4 forall BV6 : BIT6
      BV6 = MK_6 (0, 0, 0, 0, 0, 0) => S1 (BV6) = MK_4 (1, 1, 1, 0);
      BV6 = MK_6 (0, 0, 0, 0, 1, 0) => S1 (BV6) = MK_4 (0, 1, 0, 0);
      BV6 = MK_6 (0, 0, 0, 1, 0, 0) => S1 (BV6) = MK_4 (1, 1, 0, 1);
      BV6 = MK_6 (0, 0, 0, 1, 1, 0) => S1 (BV6) = MK_4 (0, 0, 0, 1);
      BV6 = MK_6 (0, 0, 1, 0, 0, 0) => S1 (BV6) = MK_4 (0, 0, 1, 0);
      BV6 = MK_6 (0, 0, 1, 0, 1, 0) => S1 (BV6) = MK_4 (1, 1, 1, 1);
      BV6 = MK_6 (0, 0, 1, 1, 0, 0) => S1 (BV6) = MK_4 (1, 0, 1, 1);
      BV6 = MK_6 (0, 0, 1, 1, 1, 0) => S1 (BV6) = MK_4 (1, 0, 0, 0);
      [... (54 lines)]
      BV6 = MK_6 (1, 1, 1, 1, 0, 1) => S1 (BV6) = MK_4 (0, 1, 1, 0);
      BV6 = MK_6 (1, 1, 1, 1, 1, 1) => S1 (BV6) = MK_4 (1, 1, 0, 1);
endtype

process S1 [INPUT, OUTPUT] : noexit :=
  INPUT ?I6:BIT6;
    OUTPUT !S1(I6);
      S1 [INPUT, OUTPUT]
endproc
```

Modeling Results

- Process calculi adapted for asynchronous circuits
- Benefits of rewrite to LNT
  - correction of minor errors (incorrect S-boxes)
  - simplified controller (7% fewer lines)
  - increased asynchronism (accept next CRYPT earlier)
  - new version of the LOTOS model
  - correction of a small bug in the LNT2LOTOS translator (support for functions called like numbers, e.g. “1”)

Analysis Challenge

- LTS size: enumeration of 64-bit input data and key
- Two approaches
  - Data abstraction
  - Environment constraints

Data Abstraction

- Abstract Bits to a singleton
  - bit vector ⇨ singleton
  - operation on bit vector ⇨ identity

- Concrete Bit

type BIT **is**
  0, 1
  with "=="
end type

- Abstract Bit

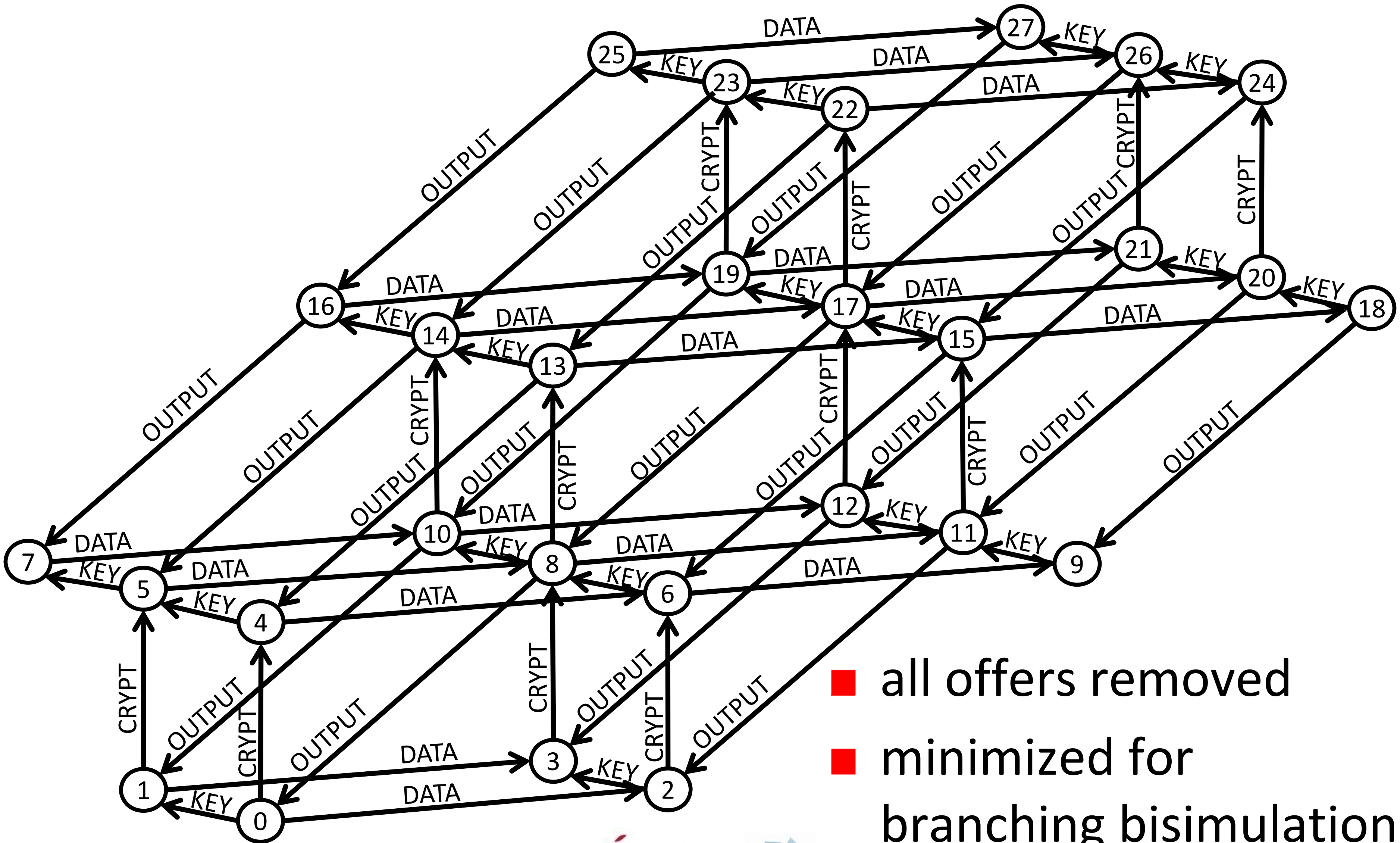
type BIT **is**
  0
  with "==“
end type
function 1 : BIT **is**
  return 0
end function

LTS Generation (Abstract Model)

	direct generation		compositional	
	LOTOS	LNT	LOTOS	LNT
states	591,914,192	167,300,852	688,422	406,192
transitions	5,542,917,498	1,500,073,686	5,122,760	2,910,530
time (minutes)	228	66	1.4	1
RAM (GB)	19.13	4.93	0.19	0.11

- Largest intermediate LTS
- Total time (without minimization for direct generation)
- Maximal RAM requirements
- Final LTS of all cases (reduced for branching bisimulation): 28 states, 78 transitions
- Measurements in August 2015 (on a Xeon® E5-2630 @2.4 GHz)

Final Abstract LTS

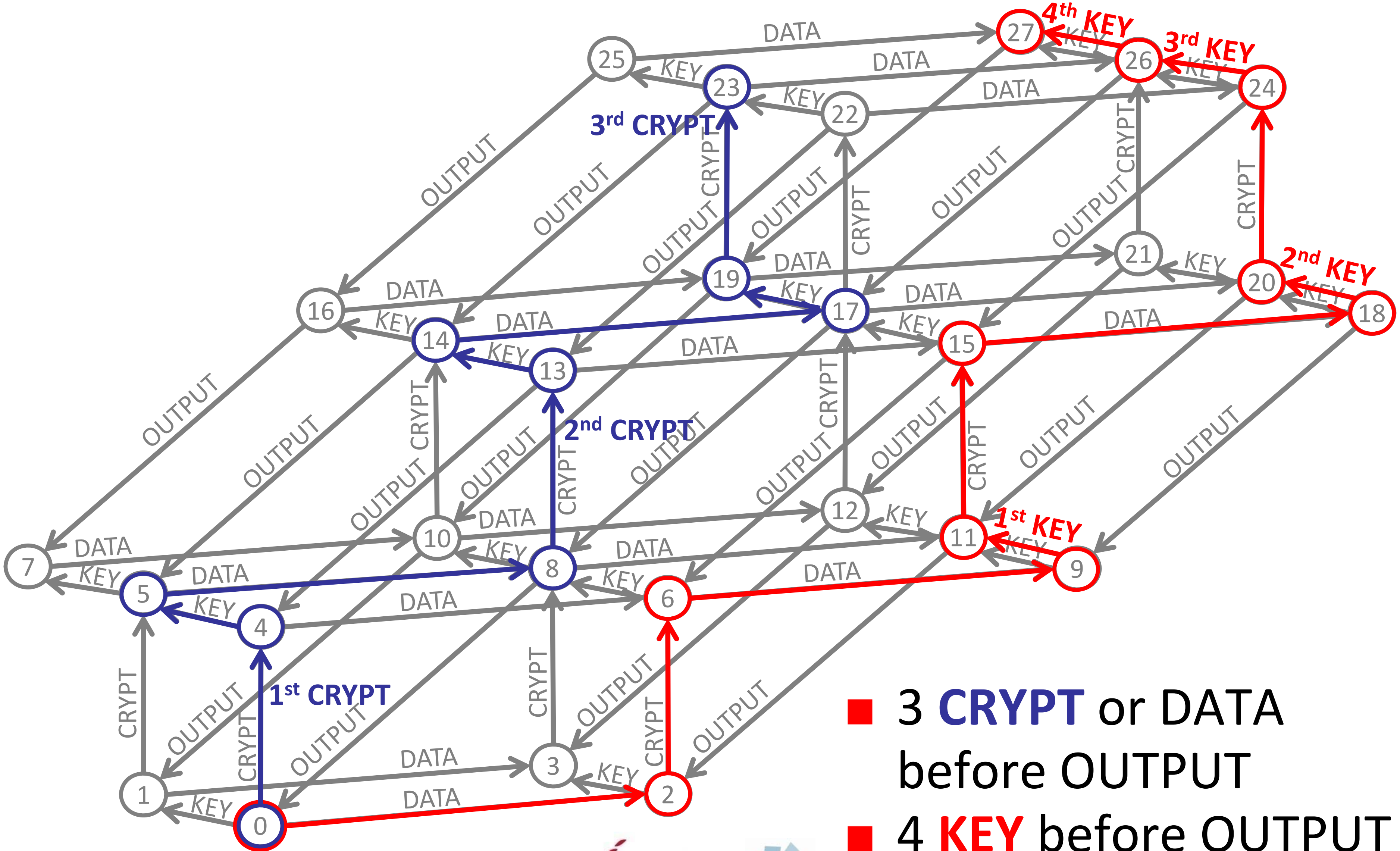


- all offers removed
- minimized for branching bisimulation

Model Checking (Abstract Model)

- Absence of deadlocks
- Triplet of inputs eventually followed by OUTPUT
  [**true*** . PARALLEL (CRYPT, DATA, KEY)]
  INEVITABLE (OUTPUT)
- Acceptance of *n* inputs *A* in advance
  - never accept more than *n* inputs *A* in advance
    [**true*** . (*A* . **not** (*A* **or** OUTPUT)*){*n*} . *A*] **false**
  - there exists an execution with *n* inputs in advance
    < **true*** . (*A* . **not** (*A* **or** OUTPUT)*){*n*-1} . *A* > **true**
  - note: *n* varies for the three inputs

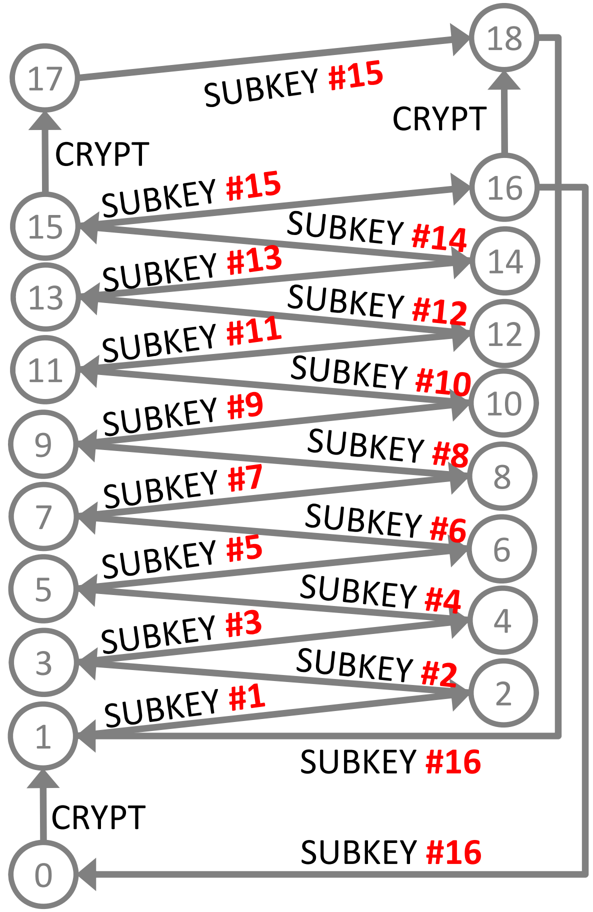
Examples of Inputs in Advance



- 3 **CRYPT** or DATA before OUTPUT
- 4 **KEY** before OUTPUT

Equivalence Checking (Abstract Model)

- Correct synchronization of data and key paths
  "sixteen SUBKEY times between two CRYPT, and
  CRYPT may not happen before fourteen SUBKEY"
- "*des_crypt_subkey.bcg*" =
  branching reduction of
  total rename
  "CRYPT.*" **->** CRYPT,
  "SUBKEY.*" **->** SUBKEY **in**
  hide DATA, OUTPUT, KEY **in**
  "*des.bcg*";
  branching comparison
  "*property_4.lnt*" **==**
  "*des_crypt_subkey.bcg*";



Direct Generation (Concrete Model)

- Environment
  - provide input: data, key, operation mode
  - check output
- Several possibilities: collateral, sequential, …
- Results
  - correct output
    [not ({ OUTPUT ... })*] INEVITABLE ({ OUTPUT ... })
  - without offers, LTS included in the final abstract LTS

Rapid Prototyping

- EXEC/CÆSAR framework
- Visible rendezvous = call to a gate function (in C)
  - data exchange with environment (read input, output)
  - accept/refuse rendezvous
- Interaction via standard input and standard output
- One line per rendezvous (LOTOS syntax)

```
CRYPT !1
DATA !0123456789abcdef
KEY !133457799bbcdff1
```

- Signal mismatch
  (unexpected rendezvous)

```
$ ./des
CRYPT !1
DATA !0123456789abcdef
KEY !133457799bbcdff1
KEY !133457799bbcdff1
CRYPT !0
DATA !85e813540f0ab405
OUTPUT !85e813540f0ab405
CRYPT !1
KEY !133457799bbcdff1
DATA !0123456789abcdef
OUTPUT !0123456789abcdef
```

Conclusion

- Challenging Benchmark
  - large, but tractable
  - experimentation with various techniques

distributed generation

- Ease of data abstraction in process calculi
- Rapid prototyping from formal models
- Complete models and verification scenario http://cadp.inria.fr/demos/demo_38