



On the security of DUKPT key derivation function

Makhonin Ilya
Akhmetzyanova Liliya
Smyshlyaev Stanislav



Motivation



Client_1



Client_2

...

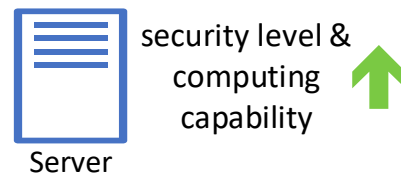
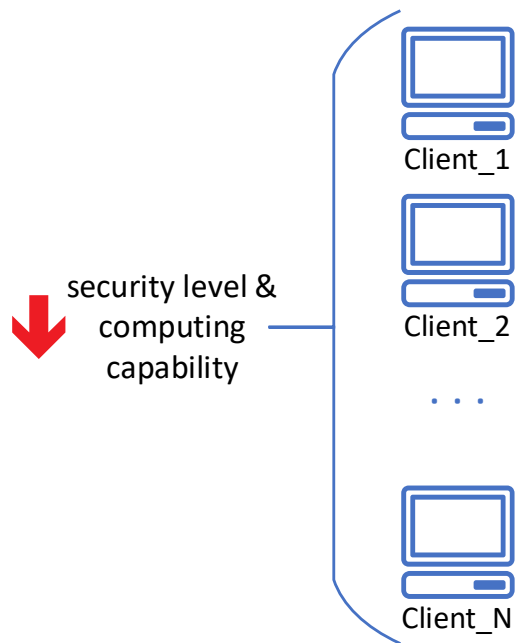


Client_N

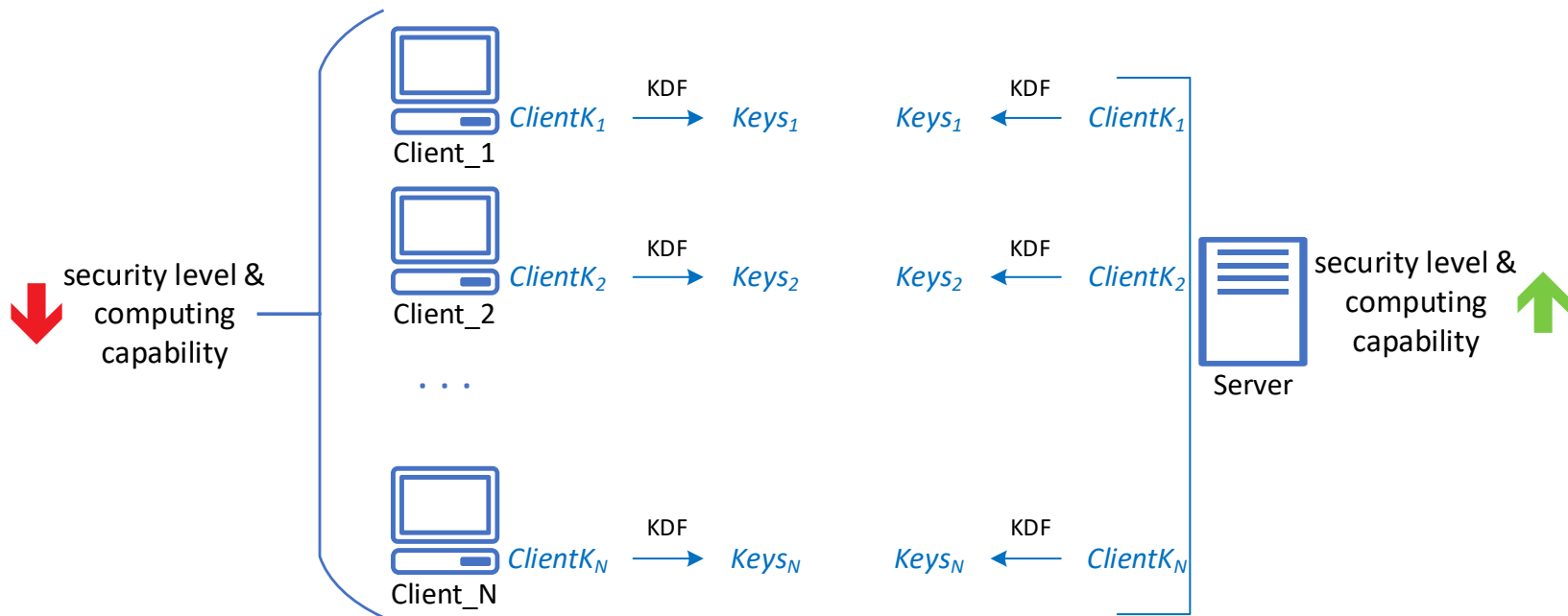


Server

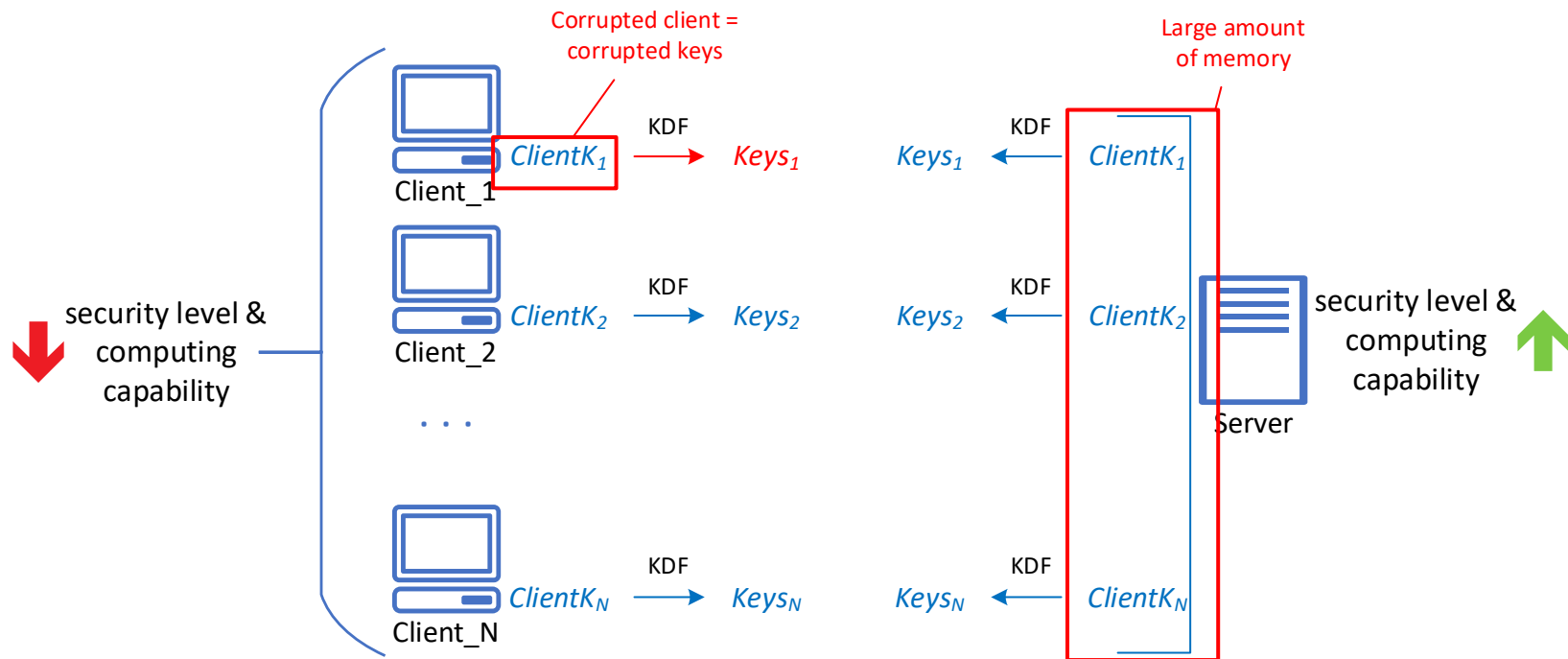
Motivation



Motivation



Motivation



Motivation

For example...

Server = Banking HSM
Client = Payment terminal

Motivation

For example...

Server = Banking HSM
Client = Payment terminal

Solution:

ANSI X9.24-3-2017 «Retail Financial Services Symmetric Key Management»
Part 3: «Derived Unique Key Per Transaction», 2017.

Motivation

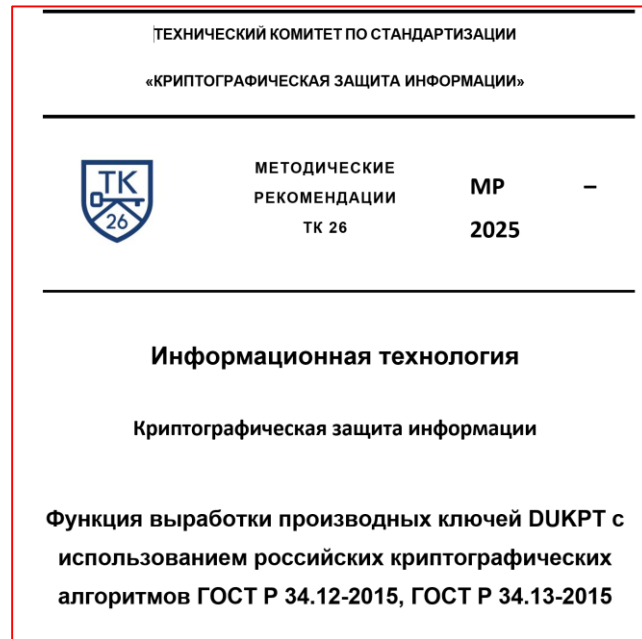
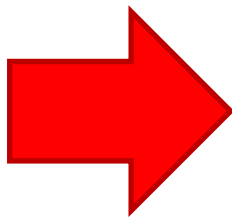
A specialized KDF is required, with two modes of operation:

1.	• storage of a single key in memory; • use on high-performance devices with a higher level of security.
2.	• rapid key derivation; • use on low-resource and less secure devices.

Motivation

A specialized KDF is required, with two modes of operation:

1.	• storage of a single key in memory; • use on high-performance devices with a higher level of security.
2.	• rapid key derivation; • use on low-resource and less secure devices.



Project of document of Technical
Committee 26

DUKPT function

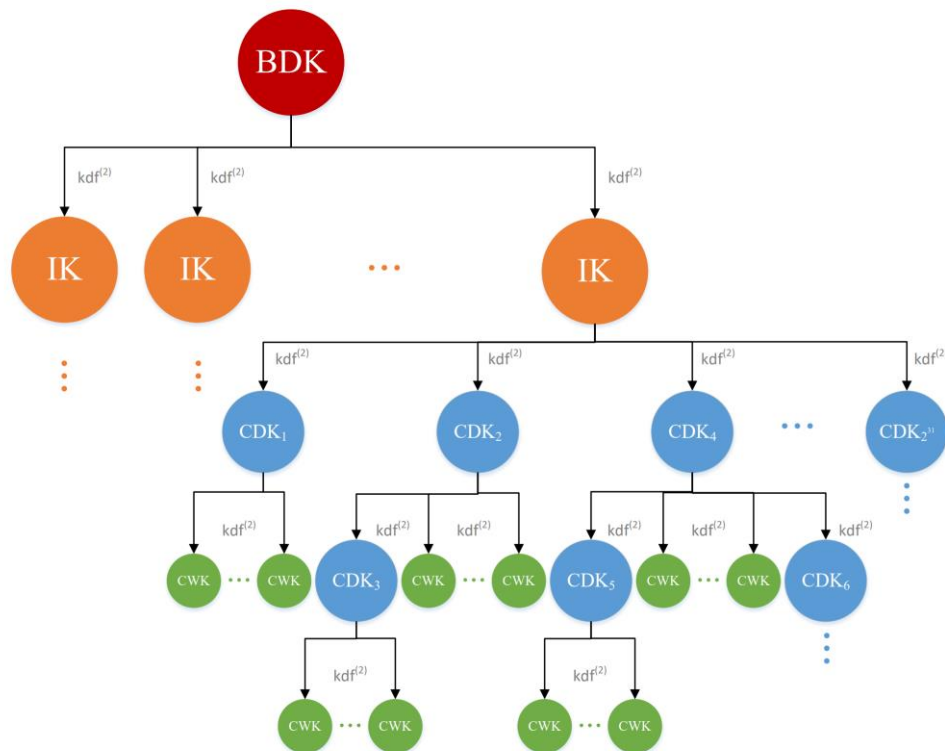
$$CWK = \text{DUKPT}(BDK, BDKID, DID, TC, U, A, L)$$

Input:

- $BDKID, DID, TC \in V_{32}$; $1 \leq wt(TC) \leq 16$ – identifiers of the base, initial, and intermediate keys, respectively;
- $U, A, L \in V_{16}$ – the usage label, algorithm label, and CWK length;

Output:

- $CWK \in V_{int(L)}$ – derived key.

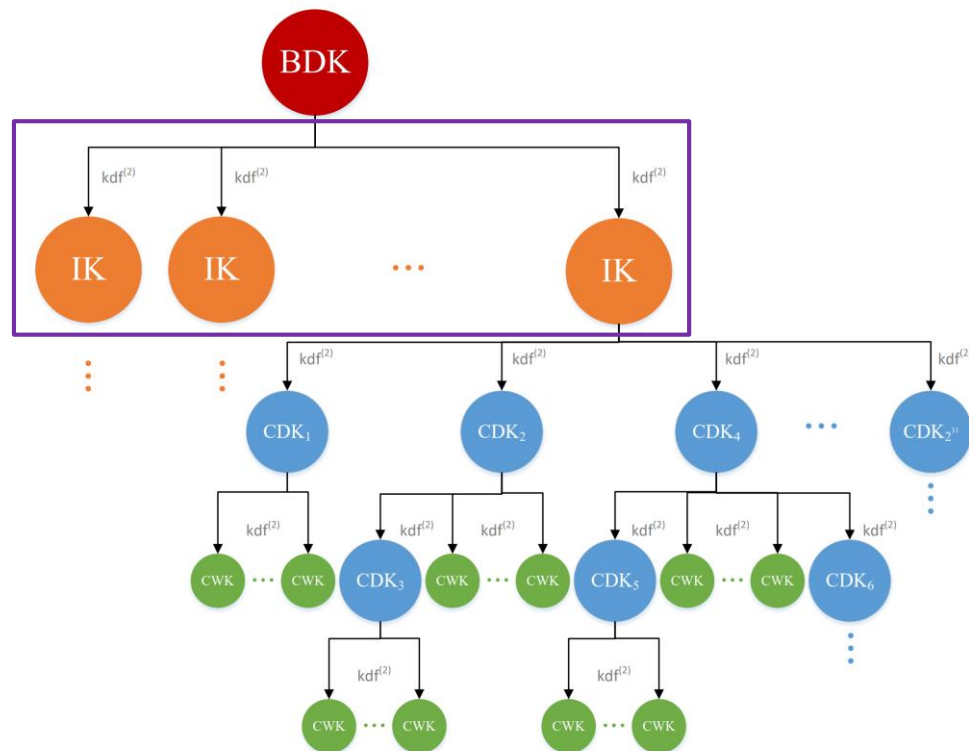


DUKPT function

$$CWK = \text{DUKPT}(BDK, BDKID, DID, TC, U, A, L)$$

Algorithm:

1. $IK = \text{DeriveIK}(BDK, BDKID, DID);$

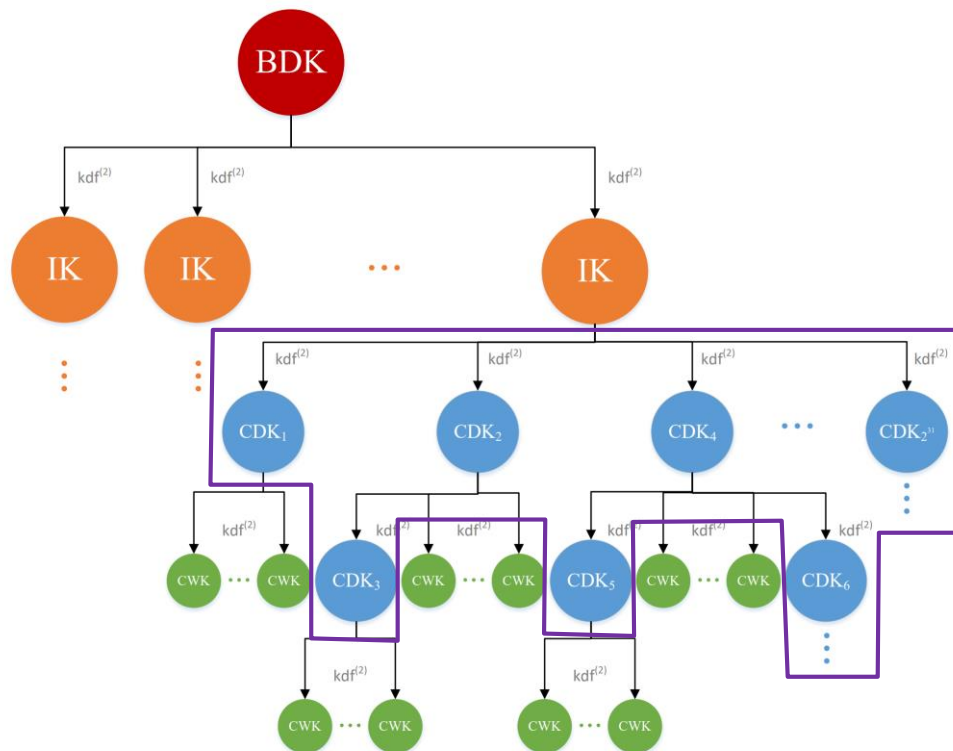


DUKPT function

$$CWK = \text{DUKPT}(BDK, BDKID, DID, TC, U, A, L)$$

Algorithm:

1. $IK = \text{DeriveIK}(BDK, BDKID, DID);$
2. $CDK = \text{DeriveCDK}(IK, DID, TC);$

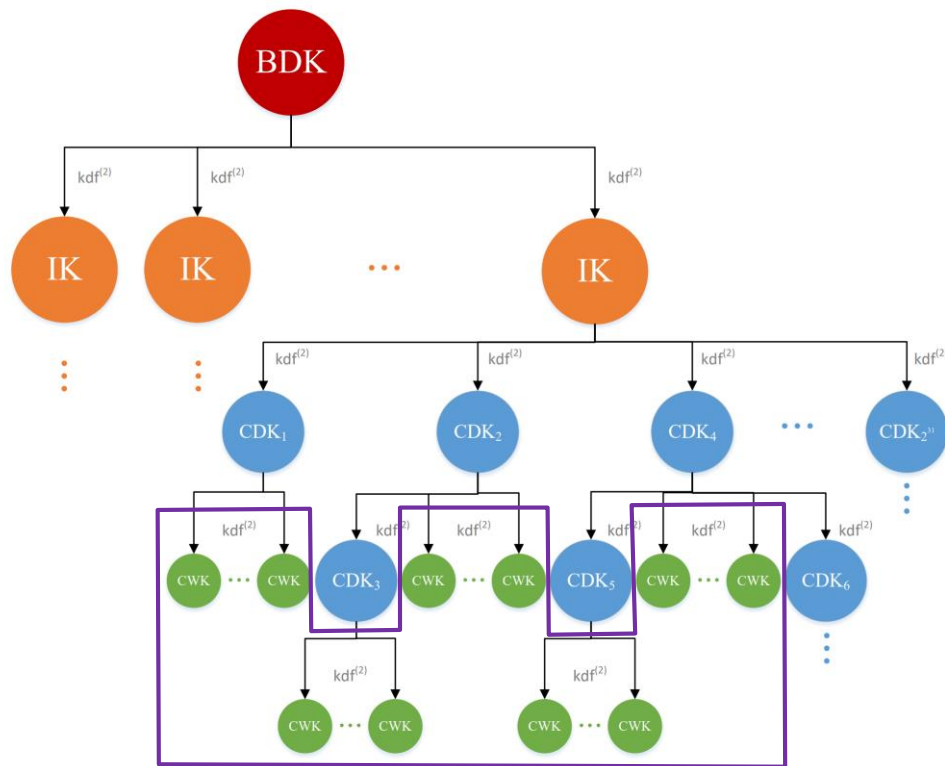


DUKPT function

$$CWK = \text{DUKPT}(BDK, BDKID, DID, TC, U, A, L)$$

Algorithm:

1. $IK = \text{DeriveIK}(BDK, BDKID, DID);$
2. $CDK = \text{DeriveCDK}(IK, DID, TC);$
3. $CWK = \text{DeriveCWK}(CDK, DID, TC, U, A, L).$



DUKPT function

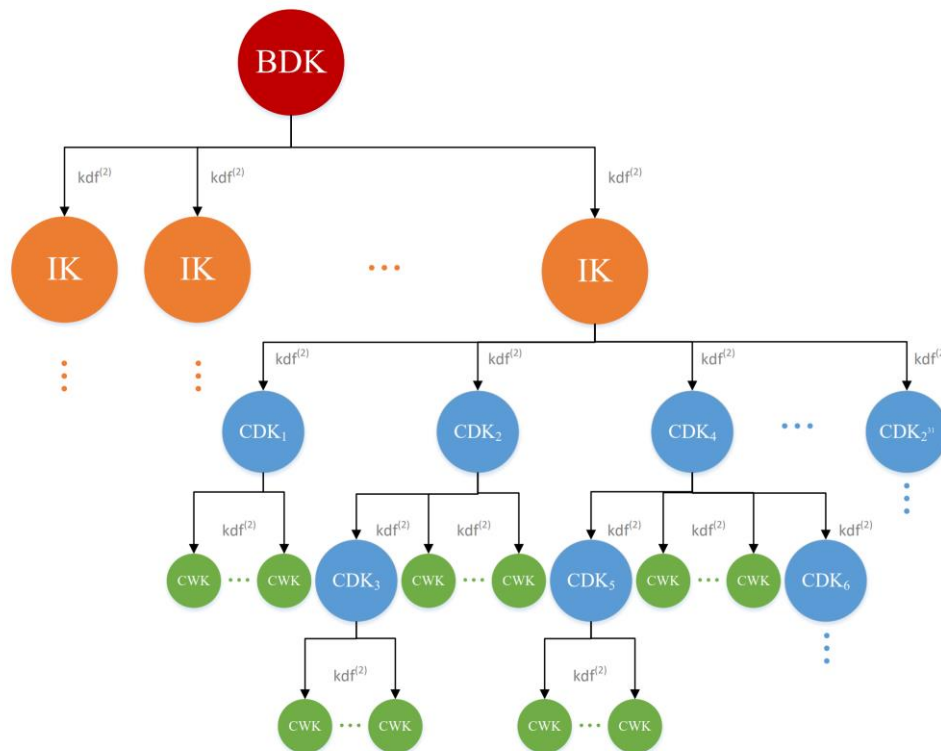
$$CWK = \text{DUKPT}(BDK, BDKID, DID, TC, U, A, L)$$

Algorithm:

1. $IK = \text{DeriveIK}(BDK, BDKID, DID);$
2. $CDK = \text{DeriveCDK}(IK, DID, TC);$
3. $CWK = \text{DeriveCWK}(CDK, DID, TC, U, A, L).$

The $kdf^{(2)}$ function defined in
R 1323565.1.022-2018.

Used with the «Kuznechik» block cipher in
CMAC.



DUKPT function

Example of derivation CDK ($TC = 6 = 0110$)

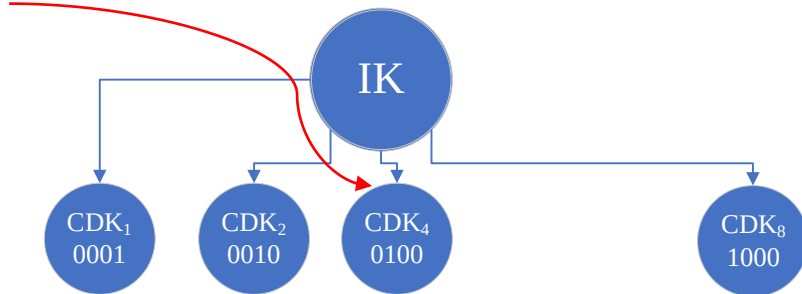


IK

DUKPT function

Example of derivation CDK ($TC = 6 = 0110$)

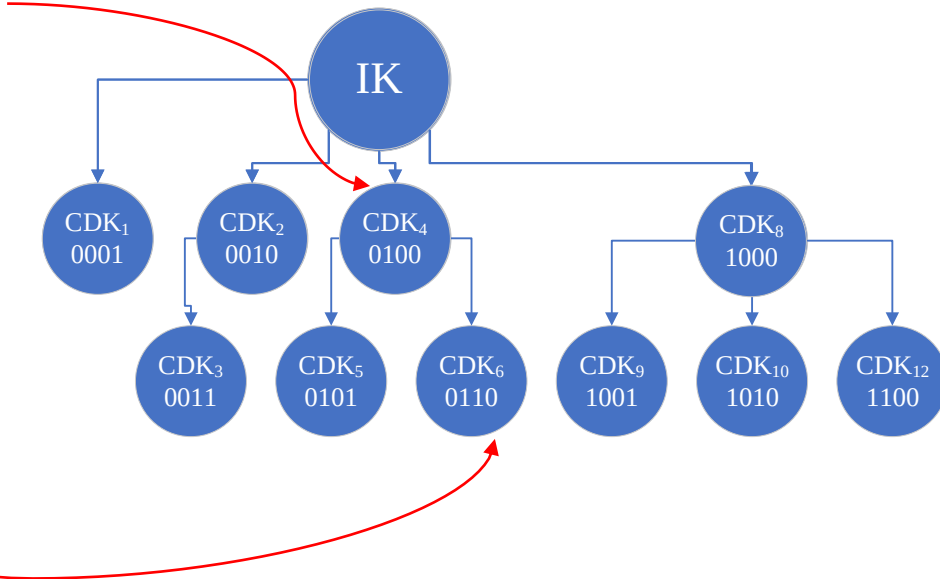
1. Derive CDK with $TC = 0100$ from IK (first bit «1»)



DUKPT function

Example of derivation CDK ($TC = 6 = 0110$)

1. Derive CDK with $TC = 0100$ from IK (first bit «1»)
2. Derive CDK with $TC = 0110$ from CDK with $TC = 0100$ (second bit «1»)



Operation modes: mode without state

Initialization:

1. Store in memory BDK , $BDKID$;

Memory:

- 1 BDK , 1 $BDKID$.

Derivation of CWK:

1. Receive DID, TC, U, A, L as input
2. $CWK =$
 $DUKPT(BDK, BDKID, DID, TC, U, A, L)$;
3. Delete DID, TC, U, A, L .

Computations:

- $\leq 18 kdf^{(2)}$.

Operation modes: mode with state

Initialization:

1. Receive $BDK, BDKID, DID$ as input;
2. Derive and store the massive $[CDK_1, CDK_2, \dots, CDK_{2^{31}}]$;
3. Delete $BDK, BDKID, IK$.

Memory:

- 1 DID , 1 TC , 32 CDK .

Derivation of CWK:

1. Receive U, A, L as input;
2. Derive CWK from CDK ;
3. Update. $state' = Update(state)$:
 1. Derive and store new CDK from used CDK
 2. Delete used CDK
4. Delete U, A, L

Computations:

- 1 $kdf^{(2)}$.

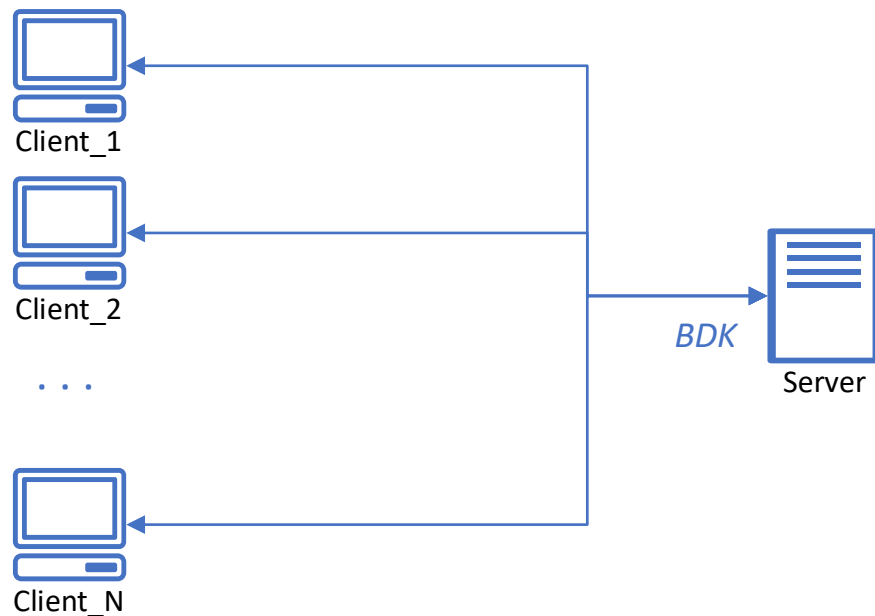
Operation modes: mode with state (example)

ptr \ TC	TC				...
	0001	0010	0011	0100	
0	0001	-	0011	-	...
1	0010	0010	-	-	
2	0100	0100	0100	0100	
3	1000	1000	1000	1000	

- ptr – index of CDK in massive;
- table cells – identifiers TC of stored CDK;
- **grey cells** will be used during next update

Operation modes

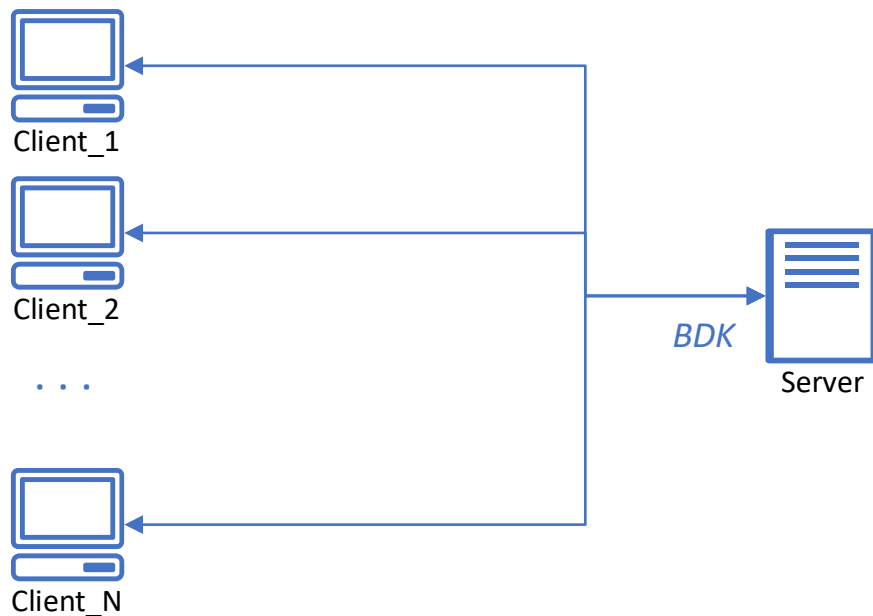
The server device operates in stateless mode.



Operation modes

The server device operates in stateless mode.

States are generated for N client devices in a distributed manner:

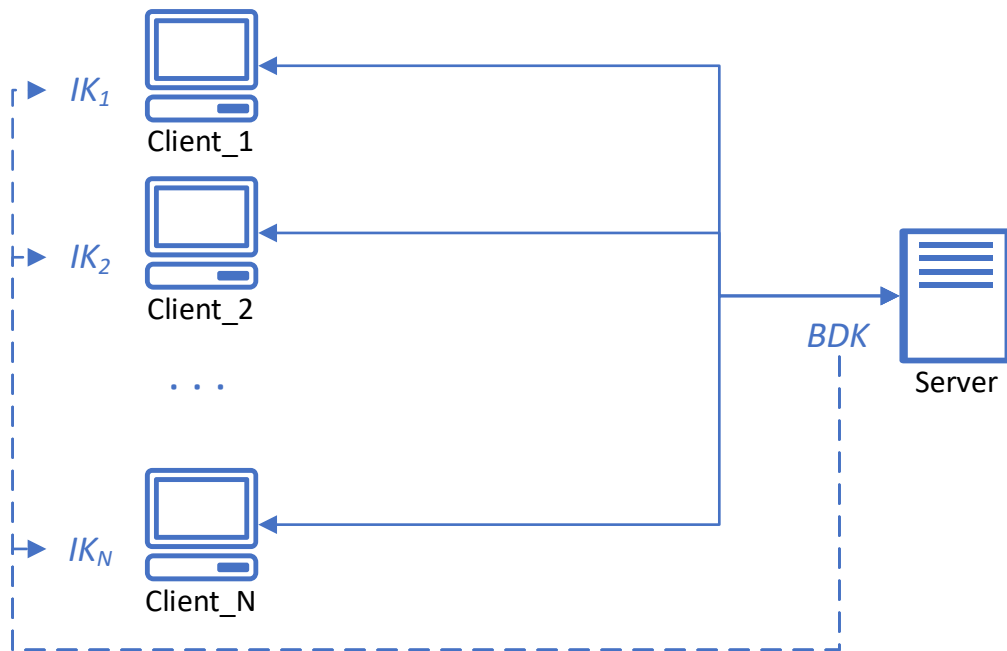


Operation modes

The server device operates in stateless mode.

States are generated for N client devices in a distributed manner:

1. IK keys are produced for N different DID .

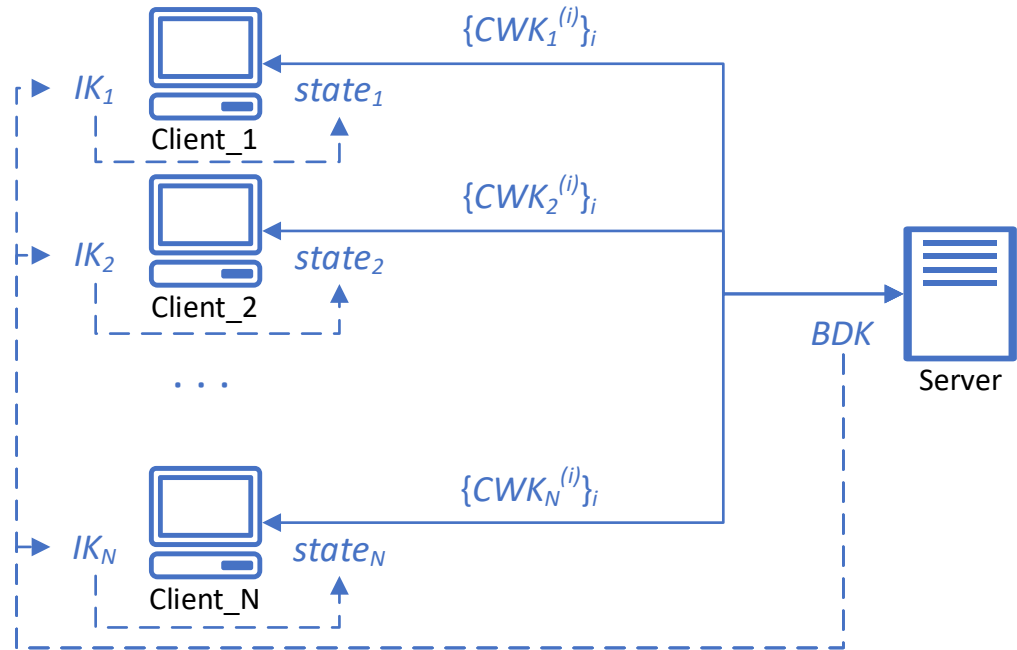


Operation modes

The server device operates in stateless mode.

States are generated for N client devices in a distributed manner:

1. IK keys are produced for N different DID .
2. For each IK, DID pair, a state is generated



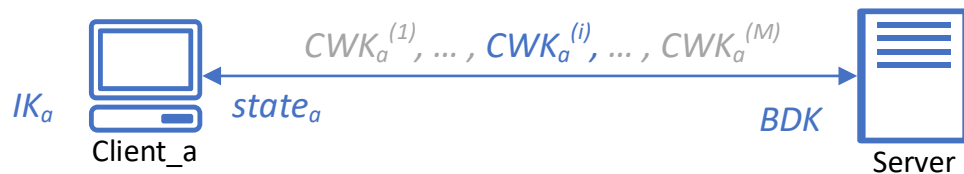
Security model

Security properties:

- Pseudorandomness of derived keys.
- Forward secrecy.

Threat:

- Distinguish CWK from a random string of the same length



Security model

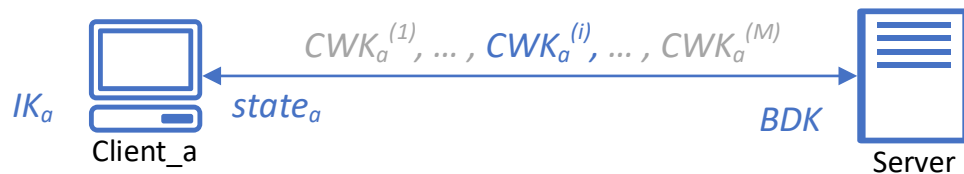
Security properties:

- Pseudorandomness of derived keys.
- Forward secrecy.

Threat:

- Distinguish CWK from a random string of the same length

Adversary capabilities:



Security model

Security properties:

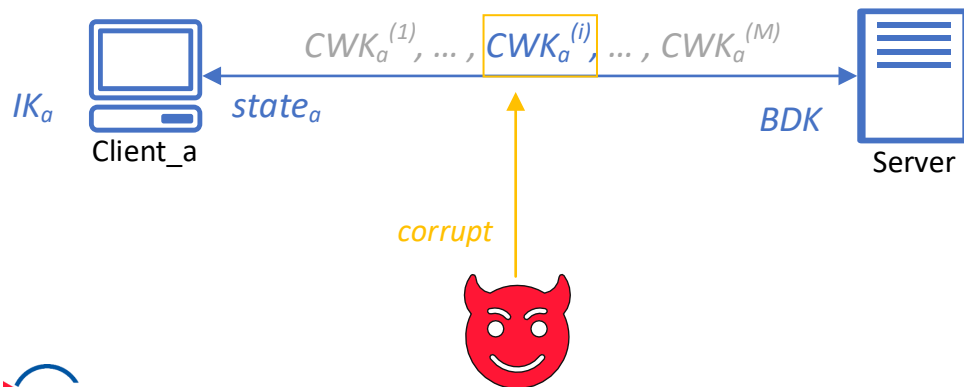
- Pseudorandomness of derived keys.
- Forward secrecy.

Threat:

- Distinguish CWK from a random string of the same length

Adversary capabilities:

1. Corrupt CWK keys for any DID, TC, U, A, L .



Security model

Security properties:

- Pseudorandomness of derived keys.
- Forward secrecy.

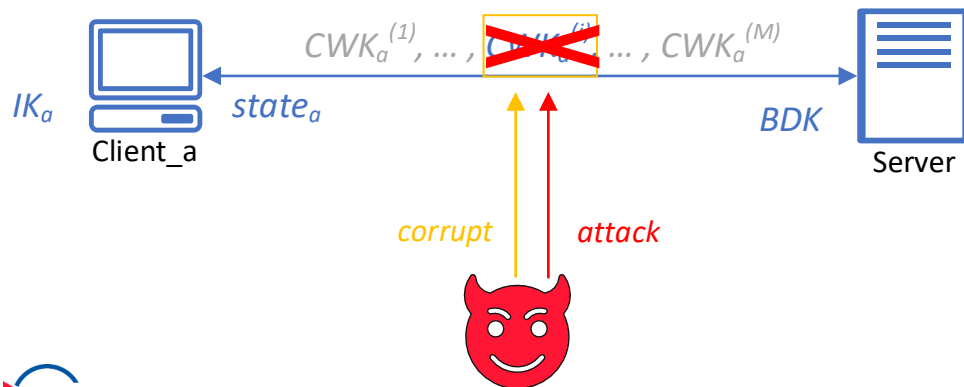
Threat:

- Distinguish CWK from a random string of the same length

Adversary capabilities:

1. Corrupt CWK keys for any DID, TC, U, A, L .

Restriction: cannot attack a corrupted CWK .



Security model

Security properties:

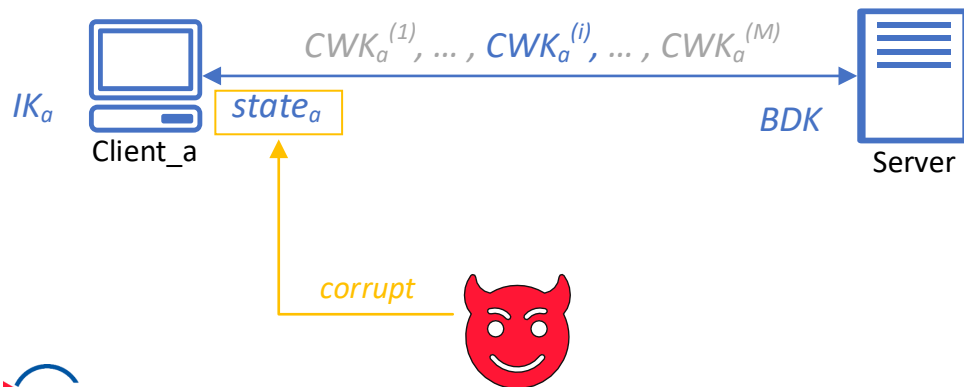
- Pseudorandomness of derived keys.
- Forward secrecy.

Threat:

- Distinguish CWK from a random string of the same length

Adversary capabilities:

1. Corrupt CWK keys for any DID, TC, U, A, L .
Restriction: cannot attack a corrupted CWK .
2. Corrupt states for any DID, TC .



Security model

Security properties:

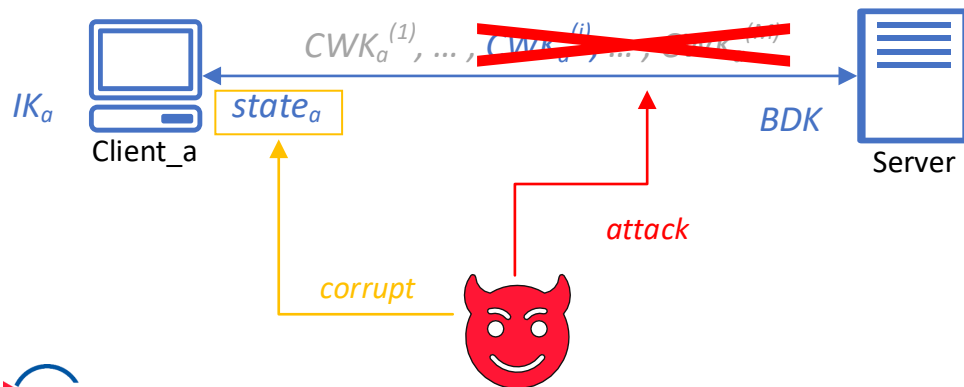
- Pseudorandomness of derived keys.
- Forward secrecy.

Threat:

- Distinguish CWK from a random string of the same length

Adversary capabilities:

1. Corrupt CWK keys for any DID, TC, U, A, L .
Restriction: cannot attack a corrupted CWK .
2. Corrupt states for any DID, TC .
Restriction: cannot attack a CWK , that can be derived from corrupted state.



Security model

Security properties:

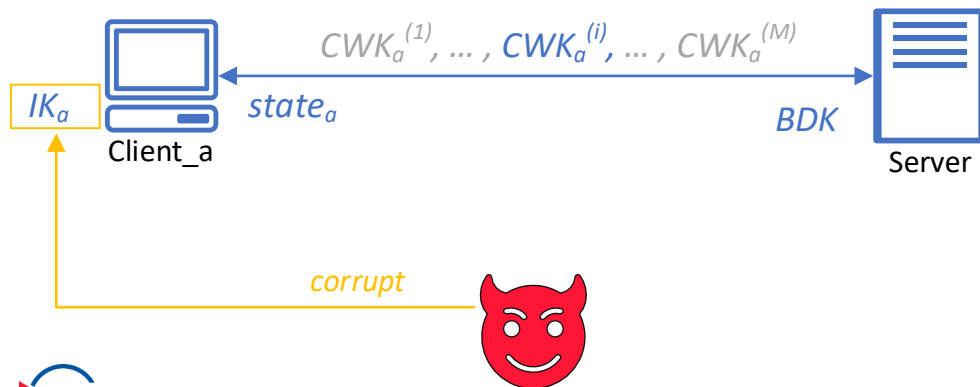
- Pseudorandomness of derived keys.
- Forward secrecy.

Threat:

- Distinguish CWK from a random string of the same length

Adversary capabilities:

1. Corrupt CWK keys for any DID, TC, U, A, L .
Restriction: cannot attack a corrupted CWK .
2. Corrupt states for any DID, TC .
Restriction: cannot attack a CWK , that can be derived from corrupted state.
3. Corrupt IK keys for any DID .



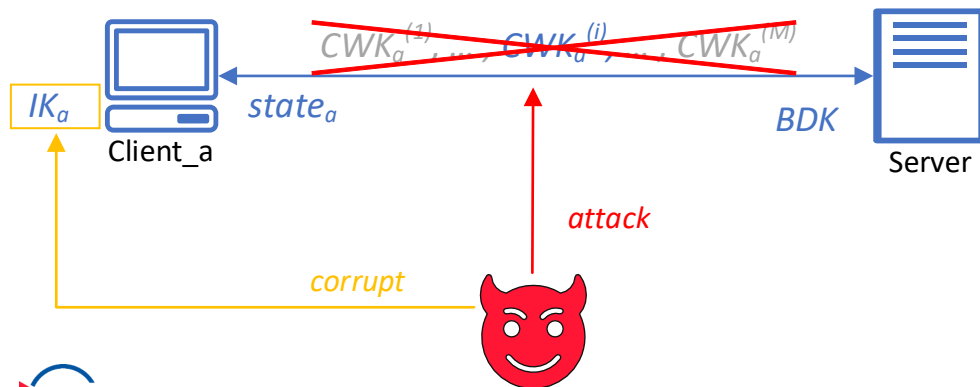
Security model

Security properties:

- Pseudorandomness of derived keys.
- Forward secrecy.

Threat:

- Distinguish CWK from a random string of the same length



Adversary capabilities:

1. Corrupt CWK keys for any DID, TC, U, A, L .
Restriction: cannot attack a corrupted CWK .
2. Corrupt states for any DID, TC .
Restriction: cannot attack a CWK , that can be derived from corrupted state.
3. Corrupt IK keys for any DID .
Restriction: cannot attack a CWK with the same DID .

Security analysis

Provable security

Concrete methods

Security analysis

Provable security

Result: Expression of the advantage of the adversary for the *DUKPT* function as a function:

$$\text{Adv}_{\text{DUKPT}}^{\text{fs-PRF}}(\mathcal{A}) = F(q_{wk}, q_{ik}, L_m, \text{Adv}_E^{\text{PRP-CPA}}(\mathcal{A}'))$$

Concrete methods

Security analysis

Provable security

Result: Expression of the advantage of the adversary for the *DUKPT* function as a function:

$$\text{Adv}_{\text{DUKPT}}^{\text{fs-PRF}}(\mathcal{A}) = F(q_{wk}, q_{ik}, L_m, \text{Adv}_E^{\text{PRP-CPA}}(\mathcal{A}'))$$

Concrete methods

Result: List of attacks on the DUKPT function with description and characteristics:

- T - Computational complexity of the attack;
- P - Probability of realization of the attack.

Provable security

Game-based approach was used.

The **Exp**(challenger) was modeled with oracles:

- *Test* – threat modeling;
- *GetWK* – corrupting *CWK*;
- *GetState* – corrupting *IK* and states.

Provable security

Game-based approach was used.

The **Exp**(challenger) was modeled with oracles:

- *Test* – threat modeling;
- *GetWK* – corrupting *CWK*;
- *GetState* – corrupting *IK* and states.

Definition. For the *DUKPT* function, the benefit of an arbitrary adversary *A* in the fs-PRF model is expressed as:

$$\text{Adv}_{DUKPT}^{\text{fs-PRF}}(\mathcal{A}) = \Pr[\mathbf{Exp}_{DUKPT}^{\text{fs-PRF}-1}(\mathcal{A}) \rightarrow 1] - \Pr[\mathbf{Exp}_{DUKPT}^{\text{fs-PRF}-0}(\mathcal{A}) \rightarrow 1]$$

fs-PRF model

$\text{Exp}_{\text{DUKPT}}^{\text{fs-PRF}-b}(\mathcal{A})$

```
1:  $BDK \xleftarrow{\mathcal{U}} V_k$ 
2:  $BDKID \leftarrow \mathcal{A}(V_{32})$ 
3:  $S_W, S_D \leftarrow \emptyset$ 
4:  $b' \leftarrow \mathcal{A}^{\text{GetWK, GetDK, TestKey}}$ 
5: return  $b'$ 
```

Oracle $\text{TestKey}(DID, TC, [U, A, L])$

```
1: if  $((DID, TC, [U, A, L]) \in S_W)$  : return FAIL
2: if  $(\exists (DID, TC') \in S_D : \text{int}(TC') \leq \text{int}(TC))$  :
3:   return FAIL
4:  $S_W \leftarrow S_W \cup \{(DID, TC, [U, A, L])\}$ 
5:
6: if  $(b = 1)$  :
7:    $K \leftarrow \text{DUKPT}(BDK, BDKID, DID, TC, U, A, L)$ 
8: else :
9:    $K \xleftarrow{\mathcal{U}} V_{\text{int}(L)}$ 
10: return  $K$ 
```

Oracle $\text{GetWK}(DID, TC, [U, A, L])$

```
1: if  $((DID, TC, [U, A, L]) \in S_W)$  : return FAIL
2: if  $(\exists (DID, TC') \in S_D : \text{int}(TC') \leq \text{int}(TC))$  :
3:   return FAIL
4:  $S_W \leftarrow S_W \cup \{(DID, TC, [U, A, L])\}$ 
5:
6:  $CWK \leftarrow \text{DUKPT}(BDK, BDKID, DID, TC, U, A, L)$ 
7: return  $CWK$ 
```

Oracle $\text{GetState}(DID, TC)$

```
1: if  $((DID, \cdot) \in S_D)$  : return FAIL
2: if  $(\exists (DID, TC', \cdot) \in S_W : \text{int}(TC') \geq \text{int}(TC))$  :
3:   return FAIL
4:  $S_D \leftarrow S_D \cup \{(DID, TC)\}$ 
5:
6: if  $(\text{int}(TC) = 0)$  :
7:    $IK \leftarrow \text{DeriveIK}(BDK, BDKID, DID)$ 
8:   return  $IK$ 
9:  $state \leftarrow \text{Init}(BDK, BDKID, DID)$ 
10: for  $i \in \overline{2, \text{int}(TC)}$  :
11:    $state' \leftarrow \text{Update}(state)$ 
12:    $state \leftarrow state'$ 
13: return  $state$ 
```

fs-PRF model

$\text{Exp}_{\text{DUKPT}}^{\text{fs-PRF}-b}(\mathcal{A})$

```
1:  $\text{BDK} \xleftarrow{\mathcal{U}} V_k$   
2:  $\text{BDKID} \leftarrow \mathcal{A}(V_{32})$   
3:  $S_W, S_D \leftarrow \emptyset$   
4:  $b' \leftarrow \mathcal{A}^{\text{GetWK, GetDK, TestKey}}$   
5: return  $b'$ 
```

Oracle $\text{TestKey}(DID, TC, [U, A, L])$

```
1: if  $((DID, TC, [U, A, L]) \in S_W)$  : return FAIL  
2: if  $(\exists (DID, TC') \in S_D : \text{int}(TC') \leq \text{int}(TC))$  :  
3:   return FAIL  
4:  $S_W \leftarrow S_W \cup \{(DID, TC, [U, A, L])\}$   
5:  
6: if  $(b = 1)$  :  
7:    $K \leftarrow \text{DUKPT}(\text{BDK}, \text{BDKID}, DID, TC, U, A, L)$   
8: else :  
9:    $K \xleftarrow{\mathcal{U}} V_{\text{int}(L)}$   
10: return  $K$ 
```

Threat modeling

Oracle $\text{GetWK}(DID, TC, [U, A, L])$

```
1: if  $((DID, TC, [U, A, L]) \in S_W)$  : return FAIL  
2: if  $(\exists (DID, TC') \in S_D : \text{int}(TC') \leq \text{int}(TC))$  :  
3:   return FAIL  
4:  $S_W \leftarrow S_W \cup \{(DID, TC, [U, A, L])\}$   
5:  
6:  $\text{CWK} \leftarrow \text{DUKPT}(\text{BDK}, \text{BDKID}, DID, TC, U, A, L)$   
7: return  $\text{CWK}$ 
```

Oracle $\text{GetState}(DID, TC)$

```
1: if  $((DID, \cdot) \in S_D)$  : return FAIL  
2: if  $(\exists (DID, TC', \cdot) \in S_W : \text{int}(TC') \geq \text{int}(TC))$  :  
3:   return FAIL  
4:  $S_D \leftarrow S_D \cup \{(DID, TC)\}$   
5:  
6: if  $(\text{int}(TC) = 0)$  :  
7:    $IK \leftarrow \text{DeriveIK}(\text{BDK}, \text{BDKID}, DID)$   
8:   return  $IK$   
9:  $\text{state} \leftarrow \text{Init}(\text{BDK}, \text{BDKID}, DID)$   
10: for  $i \in \overline{2, \text{int}(TC)}$  :  
11:    $\text{state}' \leftarrow \text{Update}(\text{state})$   
12:    $\text{state} \leftarrow \text{state}'$   
13: return  $\text{state}$ 
```

fs-PRF model

$\text{Exp}_{\text{DUKPT}}^{\text{fs-PRF}-b}(\mathcal{A})$

```
1:  $\text{BDK} \xleftarrow{\mathcal{U}} V_k$   
2:  $\text{BDKID} \leftarrow \mathcal{A}(V_{32})$   
3:  $S_W, S_D \leftarrow \emptyset$   
4:  $b' \leftarrow \mathcal{A}^{\text{GetWK, GetDK, TestKey}}$   
5: return  $b'$ 
```

Oracle $\text{TestKey}(DID, TC, [U, A, L])$

```
1: if  $((DID, TC, [U, A, L]) \in S_W)$  : return FAIL  
2: if  $(\exists (DID, TC') \in S_D : \text{int}(TC') \leq \text{int}(TC))$  :  
3:   return FAIL  
4:  $S_W \leftarrow S_W \cup \{(DID, TC, [U, A, L])\}$   
5:  
6: if  $(b = 1)$  :  
7:    $K \leftarrow \text{DUKPT}(\text{BDK}, \text{BDKID}, DID, TC, U, A, L)$   
8: else :  
9:    $K \xleftarrow{\mathcal{U}} V_{\text{int}(L)}$   
10: return  $K$ 
```

Threat modeling

Oracle $\text{GetWK}(DID, TC, [U, A, L])$

```
1: if  $((DID, TC, [U, A, L]) \in S_W)$  : return FAIL  
2: if  $(\exists (DID, TC') \in S_D : \text{int}(TC') \leq \text{int}(TC))$  :  
3:   return FAIL  
4:  $S_W \leftarrow S_W \cup \{(DID, TC, [U, A, L])\}$   
5:  
6:  $\text{CWK} \leftarrow \text{DUKPT}(\text{BDK}, \text{BDKID}, DID, TC, U, A, L)$   
7: return  $\text{CWK}$ 
```

Corrupting CWK

Oracle $\text{GetState}(DID, TC)$

```
1: if  $((DID, \cdot) \in S_D)$  : return FAIL  
2: if  $(\exists (DID, TC', \cdot) \in S_W : \text{int}(TC') \geq \text{int}(TC))$  :  
3:   return FAIL  
4:  $S_D \leftarrow S_D \cup \{(DID, TC)\}$   
5:  
6: if  $(\text{int}(TC) = 0)$  :  
7:    $\text{IK} \leftarrow \text{DeriveIK}(\text{BDK}, \text{BDKID}, DID)$   
8:   return  $\text{IK}$   
9:  $\text{state} \leftarrow \text{Init}(\text{BDK}, \text{BDKID}, DID)$   
10: for  $i \in 2, \text{int}(TC)$  :  
11:    $\text{state}' \leftarrow \text{Update}(\text{state})$   
12:    $\text{state} \leftarrow \text{state}'$   
13: return  $\text{state}$ 
```

Corrupting IK and states

fs-PRF model

$\text{Exp}_{\text{DUKPT}}^{\text{fs-PRF}-b}(\mathcal{A})$

```

1:  $\text{BDK} \xleftarrow{\mathcal{U}} V_k$ 
2:  $\text{BDKID} \leftarrow \mathcal{A}(V_{32})$ 
3:  $S_W, S_D \leftarrow \emptyset$ 
4:  $b' \leftarrow \mathcal{A}^{\text{GetWK, GetDK, TestKey}}$ 
5: return  $b'$ 
    
```

Oracle $\text{TestKey}(DID, TC, [U, A, L])$

```

1: if  $((DID, TC, [U, A, L]) \in S_W)$  : return FAIL
2: if  $(\exists (DID, TC') \in S_D : \text{int}(TC') \leq \text{int}(TC))$  :
3:   return FAIL
4:  $S_W \leftarrow S_W \cup \{(DID, TC, [U, A, L])\}$ 
5:
6: if  $(b = 1)$  :
7:    $K \leftarrow \text{DUKPT}(\text{BDK}, \text{BDKID}, DID, TC, U, A, L)$ 
8: else :
9:    $K \xleftarrow{\mathcal{U}} V_{\text{int}(L)}$ 
10: return  $K$ 
    
```

Threat modeling

Oracle $\text{GetWK}(DID, TC, [U, A, L])$

```

1: if  $((DID, TC, [U, A, L]) \in S_W)$  : return FAIL
2: if  $(\exists (DID, TC') \in S_D : \text{int}(TC') \leq \text{int}(TC))$  :
3:   return FAIL
4:  $S_W \leftarrow S_W \cup \{(DID, TC, [U, A, L])\}$ 
5:
6:  $\text{CWK} \leftarrow \text{DUKPT}(\text{BDK}, \text{BDKID}, DID, TC, U, A, L)$ 
7: return  $\text{CWK}$ 
    
```

Oracle $\text{GetState}(DID, TC)$

```

1: if  $((DID, \cdot) \in S_D)$  : return FAIL
2: if  $(\exists (DID, TC', \cdot) \in S_W : \text{int}(TC') \geq \text{int}(TC))$  :
3:   return FAIL
4:  $S_D \leftarrow S_D \cup \{(DID, TC)\}$ 
5:
6: if  $(\text{int}(TC) = 0)$  :
7:    $\text{IK} \leftarrow \text{DeriveIK}(\text{BDK}, \text{BDKID}, DID)$ 
8:   return  $\text{IK}$ 
9:  $\text{state} \leftarrow \text{Init}(\text{BDK}, \text{BDKID}, DID)$ 
10: for  $i \in \overline{2, \text{int}(TC)}$  :
11:    $\text{state}' \leftarrow \text{Update}(\text{state})$ 
12:    $\text{state} \leftarrow \text{state}'$ 
13: return  $\text{state}$ 
    
```

Accounting for trivial attacks

Corrupting CWK

Corrupting IK and states

Provable security analysis

Theorem 1. For any adversary A for the function $DUKPT$ in the fs-PRF model with computational resources at most T and parameters q_{ik}, q_{wk}, L_m , there exists an adversary $\mathcal{E}$ for block cipher E in the PRP-CPA model with computational resources at most $T' = T + 2\Delta T$ and parameter $q = \lceil \frac{L_m}{n} \rceil \max(q_{ik}, q_{wk} + 31) + 1$, for which the following relation holds:

$$\text{Adv}_{DUKPT}^{\text{fs-PRF}}(A) \leq 2 \cdot \left(q_{ik} \cdot \sum_{i=0}^{16} C_{32}^i + 1 \right) \cdot \left(\frac{4(q-1)^2 + 1}{2^n} + \text{Adv}_E^{\text{PRP-CPA}}(\mathcal{E}) \right).$$

Provable security analysis

Theorem 1. For any adversary A for the function $DUKPT$ in the fs-PRF model with computational resources at most T and parameters q_{ik}, q_{wk}, L_m , there exists an adversary $\mathcal{E}$ for block cipher E in the PRP-CPA model with computational resources at most $T' = T + 2\Delta T$ and parameter $q = \lceil \frac{L_m}{n} \rceil \max(q_{ik}, q_{wk} + 31) + 1$, for which the following relation holds:

$$\text{Adv}_{DUKPT}^{\text{fs-PRF}}(A) \leq 2 \cdot \left(q_{ik} \cdot \sum_{i=0}^{16} C_{32}^i + 1 \right) \cdot \left(\frac{4(q-1)^2 + 1}{2^n} + \text{Adv}_E^{\text{PRP-CPA}}(\mathcal{E}) \right).$$

The $\text{Insec}_E^{\text{PRP-CPA}}(T, q)$ can be estimated from above as $\frac{T}{2^k} = \frac{T}{2^{256}}$ (for the «Kuznechik»^{*}).

$$\text{Insec}_{DUKPT}^{\text{fs-PRF}}(T, q_{wk}, q_{ik}, L_m) \leq 2 \cdot \text{KeyNum} \cdot \left(\frac{4(q-1)^2 + 1}{2^n} + \frac{T'}{2^{256}} \right)$$

^{*} Иванов А.В., Матюхин Д.В., Маршалко Г.Б., Шишкин В.А., «Огонь, вода и медные трубы: 10 лет стандартизации «Кузнечика»», XXVII научно-практическая конференция «РусКрипто'2025», 2025.

Interpretation of the obtained analysis results

$$q = \left\lceil \frac{L_m}{n} \right\rceil \max(q_{ik}, q_{wk} + 31) + 1$$
$$\text{Insec}_{DUKPT}^{\text{fs-PRF}}(T, q_{wk}, q_{ik}, L_m) \leq$$
$$\underbrace{\leq 2 \cdot \text{KeyNum} \cdot \frac{4(q-1)^2 + 1}{2^n}}_{(a)} + \underbrace{2 \cdot \text{KeyNum} \cdot \frac{T'}{2^{256}}}_{(b)}.$$

Interpretation of the obtained analysis results

Collision Search in CMAC-RP

Tern (a).

Characteristics:

- $P \approx (KeyNum - (q_{ik} + 1)) \cdot \frac{Q(Q-1)}{2^{n+1}}, Q = q_{wk} \cdot \left\lceil \frac{L_m}{n} \right\rceil;$
- $T = 1.$

$$q = \left\lceil \frac{L_m}{n} \right\rceil \max(q_{ik}, q_{wk} + 31) + 1$$

$$\begin{aligned} \text{Insec}_{DUKPT}^{\text{fs-PRF}}(T, q_{wk}, q_{ik}, L_m) &\leq \\ &\leq \underbrace{2 \cdot KeyNum \cdot \frac{4(q-1)^2 + 1}{2^n}}_{\substack{(a) \\ \text{tight}}} + \underbrace{2 \cdot KeyNum \cdot \frac{T'}{2^{256}}}_{(b)}. \end{aligned}$$

Interpretation of the obtained analysis results

Collision Search in CMAC-RP

Tern (a).

Characteristics:

- $P \approx (KeyNum - (q_{ik} + 1)) \cdot \frac{Q(Q-1)}{2^{n+1}}, Q = q_{wk} \cdot \left\lceil \frac{L_m}{n} \right\rceil;$
- $T = 1.$

Key brute force of block cipher E

Tern (b).

Characteristics:

- $P = 1;$
- $T = 2^{256}.$

$$q = \left\lceil \frac{L_m}{n} \right\rceil \max(q_{ik}, q_{wk} + 31) + 1$$

$$\begin{aligned} \text{Insec}_{DUKPT}^{\text{fs-PRF}}(T, q_{wk}, q_{ik}, L_m) &\leq \\ &\leq \underbrace{2 \cdot KeyNum \cdot \frac{4(q-1)^2 + 1}{2^n}}_{(a) \text{ tight}} + \underbrace{2 \cdot KeyNum \cdot \frac{T'}{2^{256}}}_{(b) \text{ not tight??}} \end{aligned}$$

Time-Memory-Trade-off attack

Time-Memory-Trade-off attack

Algorithm:

1. For a fixed m form a table of M pairs $k_i, E_{k_i}(m)$;

k_1	m	$E_{k_1}(m)$
...		...
k_M		$E_{k_M}(m)$

Time-Memory-Trade-off attack

Algorithm:

1. For a fixed m form a table of M pairs $k_i, E_{k_i}(m)$;
2. Obtain for T different keys k'_j the values $E_{k'_j}(m)$;

k_1	m	$E_{k_1}(m)$
...		...
k_M		$E_{k_M}(m)$

$E_{k'_1}(m)$
...
$E_{k'_M}(m)$

Time-Memory-Trade-off attack

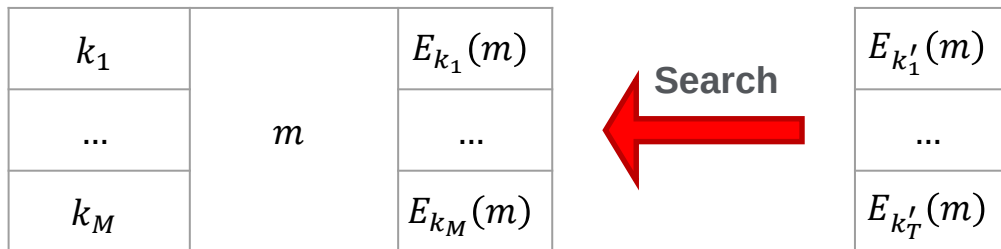
Algorithm:

1. For a fixed m form a table of M pairs $k_i, E_{k_i}(m)$;
2. Obtain for T different keys k'_j the values $E_{k'_j}(m)$;
3. Search the table for the values of $E_{k'_j}(m)$.

$$E_{k'_j}(m) = E_{k_i}(m) \Rightarrow k'_j = k_i.$$

Characteristics:

- $P = 1$;
- $T = \frac{2^k}{M}$.



Time-Memory-Trade-off attack

Algorithm:

1. For a fixed m form a table of M pairs $k_i, E_{k_i}(m)$;
2. Obtain for T different keys k'_j the values $E_{k'_j}(m)$;
3. Search the table for the values of $E_{k'_j}(m)$.

$$E_{k'_j}(m) = E_{k_i}(m) \Rightarrow k'_j = k_i.$$

Characteristics:

- $P = 1$;
- $T = \frac{2^k}{M}$.

k_1	m	$E_{k_1}(m)$
...		...
k_M		$E_{k_M}(m)$



~~$E_{k'_1}(m)$~~
~~...~~
~~$E_{k'_T}(m)$~~

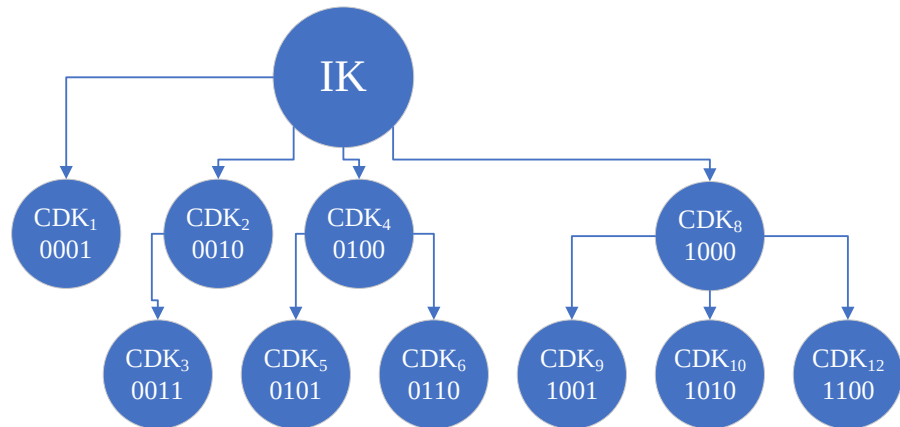
DUKPT:

$E_{k'_1}(m_1)$
...
$E_{k'_T}(m_T)$

$$m_i \neq m_j$$

«Hidden» method

Key brute force of block cipher by truncated set



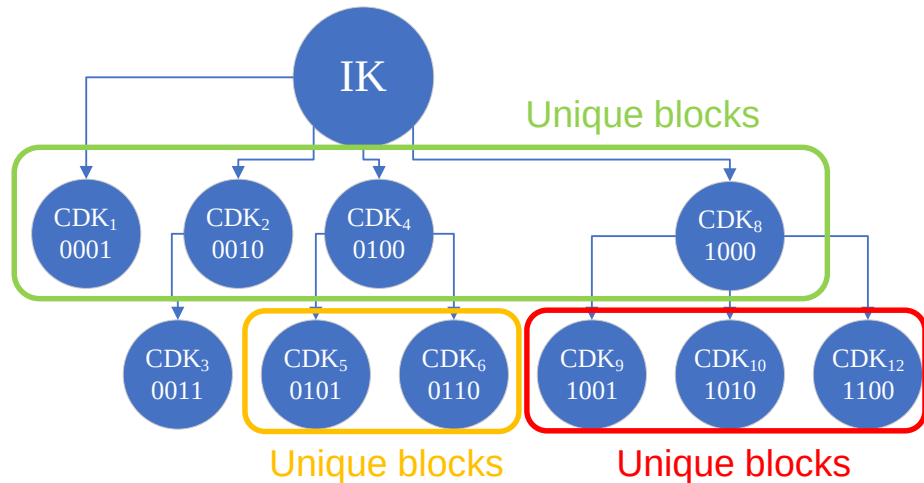
«Hidden» method

Key brute force of block cipher by truncated set

Main idea:

Keys of the same level derived from one key have **unique** blocks.

Attack one target key, blocks of other keys are excluded



«Hidden» method

Key brute force of block cipher by truncated set

Main idea:

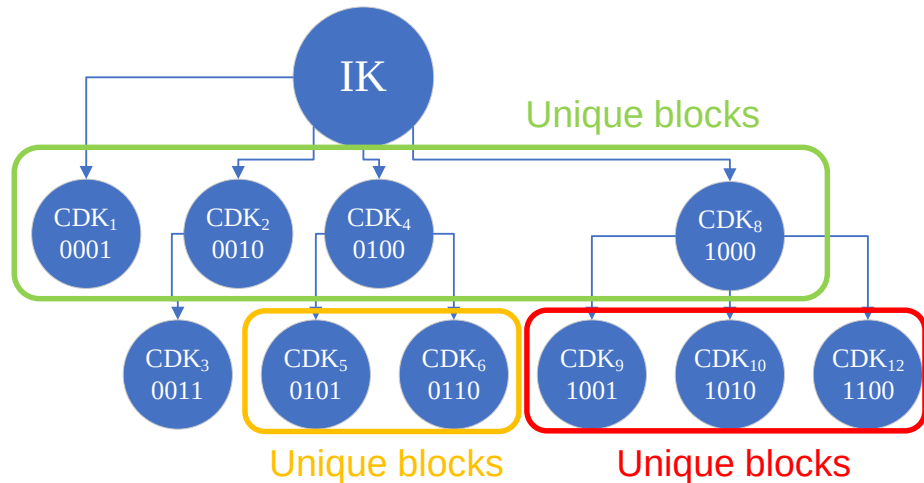
Keys of the same level derived from one key have **unique** blocks.

Attack one target key, blocks of other keys are excluded

Characteristics:

- $T \approx (2^{128} - N_B)(2^{128} - N_B - 1);$
- $P = 1;$

where $N_B = \max(2 \cdot (q_{ik} - 1), 60 + q_{wk} \cdot \left\lceil \frac{L_m}{128} \right\rceil).$



«Hidden» method

Key brute force of block cipher by truncated set

Main idea:

Keys of the same level worked out by one key have **unique** blocks.

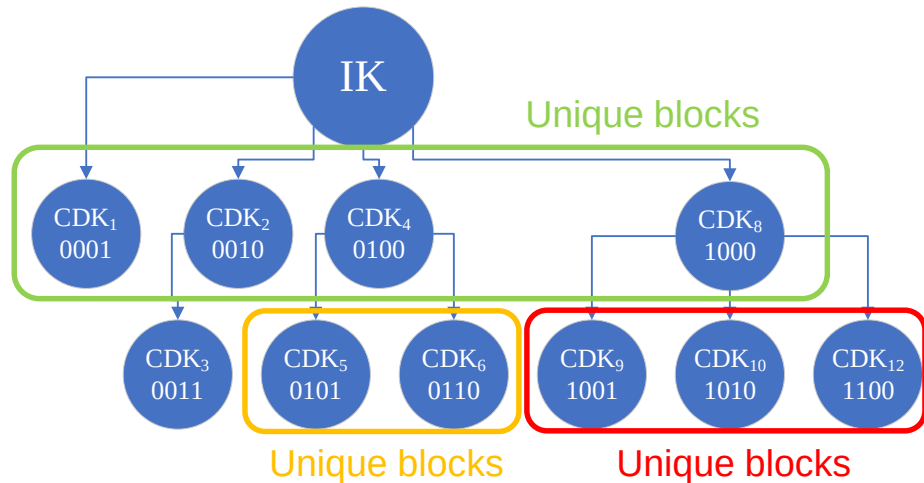
Attack one target key, blocks of other keys are excluded

Characteristics:

- $T \approx (2^{128} - N_B)(2^{128} - N_B - 1);$
- $P = 1;$

where $N_B = \max(2 \cdot (q_{ik} - 1), 60 + q_{wk} \cdot \left\lceil \frac{L_m}{128} \right\rceil).$

✓ Multiplier (c).



$$\begin{aligned} \text{Insec}_{DUKPT}^{\text{fs-PRF}}(T, q_{wk}, q_{ik}, L_m) &\leq \\ &\leq 2 \cdot \text{KeyNum} \cdot \underbrace{\left(\frac{4(q-1)^2 + 1}{2^n} + \frac{T'}{2^{256}} \right)}_{(c)} \end{aligned}$$

Security estimates

Parameters:

- Number of *CWK* keys derived at a time:
 $q_{wk} = 2^{10}$;
- Number of clients: $q_{ik} = 2^{23}$;
- Length of one *CWK*: $L_m = 2^{16} - 1$;
- Number of tree keys, excluding *CWK* keys: $KeyNum \approx 2^{54}$

Attack	T	P
Key brute force of block cipher E	2^{256}	1
Collision Search in $CMAC - RP$	1	2^{-37}
Key brute force of block cipher E by truncated set	$2^{255,9}$	1

$$\text{Insec}_{DUKPT}^{\text{fs-PRF}}(T, q_{wk}, q_{ik}, L_m) \leq 2^{-35} + \frac{T}{2^{200}}.$$

Results

1. The analysis of the *DUKPT* function is performed using complexity-theoretic approach and analysis of concrete methods.
2. The results of the analysis by both approaches show the security of the *DUKPT* function.
3. The characteristics of the attacks are not inconsistent with the Insec estimation.



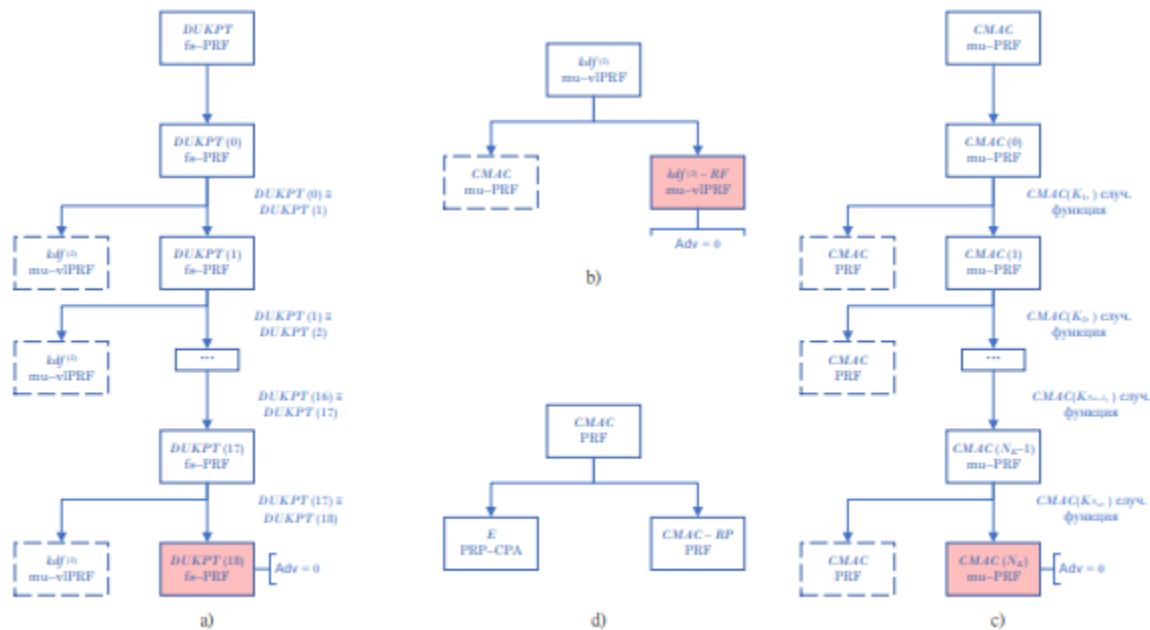


**Thank you for your
attention!**



Proof. (sketch) Доказательство теоремы состоит из четырех шагов. На каждом шаге мы идеализируем определенный внутренний элемент функции *DUKPT* путем замены на идеальный «аналог». Например, заменяем определенную ключевую функцию на функцию с такими же областью определения и областью значения, выбранную случайно равномерно. Под «стоимостью» идеализации подразумеваем преимущество наиболее эффективного нарушителя для задачи отличия исходного элемента от идеализированного.

Шаги доказательства представлены на рисунке 2.



Шаг 1. На шаге 1, на рисунке 2а (Лемма 1) мы поэтапно идеализируем функцию $kdf^{(2)}$ в функции $DUKPT$, заменяя ее на случайную функцию с соответствующими областью определения и областью значения.

На данном шаге поочередно идеализируется каждый уровень дерева, начиная с первого. То есть, при идеализации i -го уровня, ключи данного уровня дерева меняются на случайные, что идентично идеализации функции $kdf^{(2)}$ при выработке ключей i -го уровня. Функцию $DUKPT$, в дереве ключей которой все ключи уровней от 0 до i заменены на случайные строки соответствующей длины, обозначим через $DUKPT(i)$. Таким образом, на каждом этапе идеализации мы переходим от анализа функции $DUKPT(i)$ к анализу $DUKPT(i + 1)$. Таким способом выполняется идеализация до функции $DUKPT(18)$, в которой все ключи дерева случайные. Соответственно, преимущество любого нарушителя для функции $DUKPT(18)$ в модели fs-PRF будет равно 0.

«Стоимостью» идеализации на каждом этапе является преимущество наилучшего нарушителя в модели, которая рассматривает стойкость функции $kdf^{(2)}$ к угрозе отличия результата ее работы на наборе различных ключей от случайной строки той же длины (функция $kdf^{(2)}$ в модели mu-vlPRF, см. Определение 3, приложение С). Оценка преимущества нарушителя в данной модели выполняется на шаге 2. Количество ключей в модели определяется исходя из того, на каком этапе идеализации дерева преимущество нарушителя в данной модели было получено в качестве «стоимости». При идеализации i -го уровня количество ключей функции $kdf^{(2)}$ в модели mu-vlPRF будет равно количеству ключей $(i - 1)$ -го уровня дерева (кроме ключей CWK).

На данном шаге получаем следующее соотношение:

$$\text{Adv}_{DUKPT}^{\text{fs-PRF}}(\mathcal{A}) \leq 2 \cdot \sum_{i=1}^{18} \text{Adv}_{kdf^{(2)}}^{\text{mu-vlPRF}}(\mathcal{B}_i).$$



Шаг 2. На шаге 2, на рисунке 2b (Лемма 2) мы в каждой функции $kdf^{(2)}$ на наборе ключей идеализируем функцию $CMAC$, заменяя ее на случайную функцию с соответствующими областью определения и областью значения.

В результате идеализации мы получим функцию $kdf^{(2)}$ со случайной функцией (обозначим такую функцию через $kdf^{(2)}-RF$). Так как выход функции $kdf^{(2)}-RF$ представляет собой конкатенацию выходов случайной функции на разных входных значениях, преимущество любого нарушителя для функции $kdf^{(2)}-RF$ в модели μ -vIPRF будет равно 0. «Стоимостью» идеализации является преимущество наилучшего нарушителя в модели, которая рассматривает стойкость функции $CMAC$ к угрозе отличия результата ее работы на наборе различных ключей от случайной строки той же длины (функция $CMAC$ в модели μ -PRF, см. Определение 4, приложение C). Оценка преимущества нарушителя в данной модели выполняется на шаге 3.

На данном шаге получаем следующее соотношение:

$$\text{Adv}_{kdf^{(2)}}^{\mu\text{-vIPRF}}(\mathcal{B}) \leq \text{Adv}_{CMAC}^{\mu\text{-PRF}}(\mathcal{C}).$$



Шаг 3. На шаге 3, на рисунке 2с (Лемма 3) мы поэтапно идеализируем функцию $CMAC$ на определенном наборе ключей, заменяя ее на случайную функцию с соответствующими областью определения и областью значения.

Для набора ключей $KeySet = \{K_i\}_{i=1}^{N_K}$ в модели mu -PRF, введем соответствующий набор функций $\{CMAC(K_i, \cdot)\}_{i=1}^{N_K}$. Обозначим через $CMAC(m)$ такое семейство функций $CMAC$, в котором функции $CMAC(K_1, \cdot), \dots, CMAC(K_m, \cdot)$ заменены на случайные. На первом этапе, идеализируя функцию $CMAC(K_1, \cdot)$, мы переходим к анализу семейства функций $CMAC(1)$ в модели mu -PRF, и так далее. После идеализации последней функции $CMAC(K_{N_K}, \cdot)$ все функции набора будут заменены на случайные. Преимущество нарушителя для семейства функций $CMAC(N_K)$ в модели mu -PRF будет равно 0, поскольку нарушитель в ответ на любой запрос к оракулу будет получать всегда результат работы случайной функции.

«Стоимостью» идеализации на каждом этапе является преимущество наилучшего нарушителя в модели, которая рассматривает стойкость функции $CMAC$ к угрозе отличия результата ее работы от случайной строки той же длины (функция $CMAC$ в модели PRF, см. Определение 5, приложение С). Оценка преимущества нарушителя в данной модели выполняется на шаге 4.

На данном шаге получаем следующее соотношение:

$$Adv_{CMAC}^{mu-PRF}(\mathcal{C}) \leq N_K Adv_{CMAC}^{PRF}(\mathcal{D}).$$



Шаг 4. На шаге 4, на рисунке 2d (Лемма 4) мы идеализируем блочный шифр, заменяя его на случайную подстановку на соответствующем множестве значений.

В результате идеализации мы получим функцию $CMAC$, в которой в качестве блочного шифра используется случайная подстановка (обозначим такую функцию через $CMAC-RP$). На данном шаге показано, что любые методы, нацеленные на отличие результата работы функции $CMAC$ от случайной строки той же длины относятся к одной из следующих групп методов:

- Методы, нацеленные на отличие результата работы функции $CMAC-RP$ от случайной строки той же длины (функция $CMAC - RP$ в модели PRF).
- Методы, нацеленные на отличие блочного шифра E на случайном ключе от случайной подстановки (блочный шифр E в модели PRP-CPA).

«Стоимостью» данной идеализации является преимущество наилучшего нарушителя в модели, которая рассматривает стойкость блочного шифра E на случайном ключе к угрозе его отличия от случайной подстановки (блочный шифр E в модели PRP-CPA).

На данном шаге получаем следующее соотношение:

$$\text{Adv}_{CMAC}^{\text{PRF}}(\mathcal{D}) \leq \frac{(4(q-1)^2 + 1)}{2^n} + \text{Adv}_E^{\text{PRP-CPA}}(\mathcal{E}).$$



Appendix mu-vIPRF model

$\mathbf{Exp}_{kdf^{(2)}}^{\text{mu-vIPRF}-b}(\mathcal{A})$

```
1 :  if ( $b = 1$ ) :  
2 :       $KeySet \leftarrow \left\{ K_i : K_i \xleftarrow{\mathcal{U}} V_{256} \right\}_{i=1}^{N_K}$   
3 :   $S \leftarrow \emptyset$   
4 :   $b' \leftarrow \mathcal{A}^{PRF, Test}$   
5 :  return  $b'$ 
```

Oracle $Test([L, P, U, A], i)$

```
1 :  if ( $([L, P, U, A], i) \in S$ ) : return FAIL  
2 :   $S \leftarrow S \cup \{([L, P, U, A], i)\}$   
3 :  if ( $b = 1$ ) :  
4 :       $out \leftarrow kdf^{(2)}(K_i, L, P, U, A)$   
5 :  else :  
6 :       $out \xleftarrow{\mathcal{U}} V_L$   
7 :  return  $out$ 
```

$$\text{Adv}_{kdf^{(2)}}^{\text{mu-vIPRF}}(\mathcal{A}) = \Pr[\mathbf{Exp}_{kdf^{(2)}}^{\text{mu-vIPRF}-1}(\mathcal{A}) \rightarrow 1] - \Pr[\mathbf{Exp}_{kdf^{(2)}}^{\text{mu-vIPRF}-0}(\mathcal{A}) \rightarrow 1]$$

Appendix mu-PRF model

$\text{Exp}_{CMAC}^{\text{mu-PRF}-b}(\mathcal{A})$

```
1:  if ( $b = 1$ ) :  
2:       $KeySet \leftarrow \left\{ K_i : K_i \xleftarrow{\mathcal{U}} V_{256} \right\}_{i=1}^{N_K}$   
3:   $S \leftarrow \emptyset$   
4:   $b' \leftarrow \mathcal{A}^{Test}$   
5:  return  $b'$ 
```

Oracle $Test(data, i)$

```
1:  if  $((data, i) \in S)$  : return FAIL  
2:   $S \leftarrow S \cup \{(data, i)\}$   
3:  if ( $b = 1$ ) :  
4:       $out \leftarrow CMAC(K_i, data)$   
5:  else :  
6:       $out \xleftarrow{\mathcal{U}} V_n$   
7:  return  $out$ 
```

$$\text{Adv}_{CMAC}^{\text{mu-PRF}}(\mathcal{A}) = \Pr[\text{Exp}_{CMAC}^{\text{mu-PRF}-1}(\mathcal{A}) \rightarrow 1] - \Pr[\text{Exp}_{CMAC}^{\text{mu-PRF}-0}(\mathcal{A}) \rightarrow 1]$$

Appendix PRF model

$\mathbf{Exp}_{CMAC}^{\text{PRF}-b}(\mathcal{A})$

```
1: if ( $b = 1$ ) :  
2:    $K \xleftarrow{\mathcal{U}} V_k$   
3:    $S \leftarrow \emptyset$   
4:    $b' \leftarrow \mathcal{A}^{GetVal}$   
5:   return  $b'$ 
```

$\text{Oracle } Test(data)$

```
1: if ( $data \in S$ ) : return FAIL  
2:    $S \leftarrow S \cup \{data\}$   
3:   if ( $b = 1$ ) :  
4:      $out \leftarrow CMAC(K, data)$   
5:   else :  
6:      $out \xleftarrow{\mathcal{U}} V_n$   
7:   return  $out$ 
```

$$\text{Adv}_{CMAC}^{\text{PRF}}(\mathcal{A}) = \Pr[\mathbf{Exp}_{CMAC}^{\text{PRF}-1}(\mathcal{A}) \rightarrow 1] - \Pr[\mathbf{Exp}_{CMAC}^{\text{PRF}-0}(\mathcal{A}) \rightarrow 1]$$