A Forward-Secure Symmetric-Key Derivation Protocol - How to Improve Classical DUKPT

Eric Brier and Thomas Peyrin

Ingenico and NTU

Asiacrypt 2010

Singapore - December 6, 2010

Outline

Introduction and motivation

Derived Unique Key Per Transaction (DUKPT)

Optimal-DUKPT

Comparison and Optimality

Outline

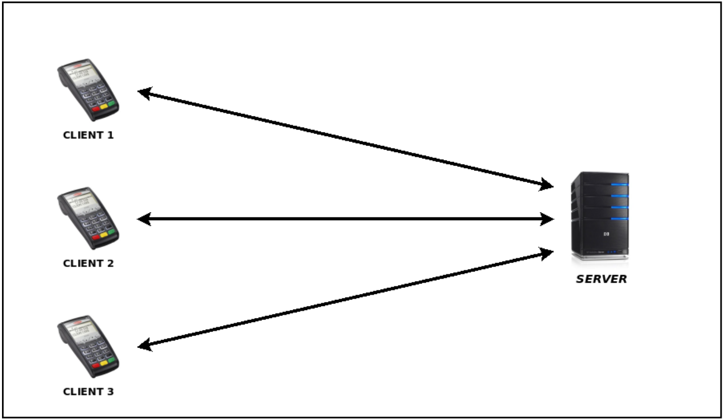
Introduction and motivation

Derived Unique Key Per Transaction (DUKPT)

Optimal-DUKPT

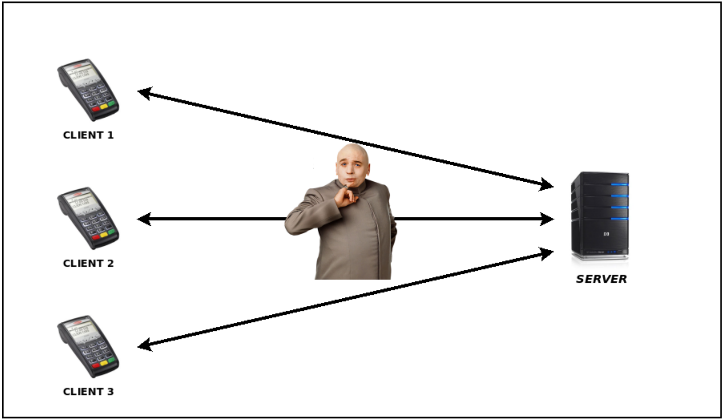
Comparison and Optimality

Motivation



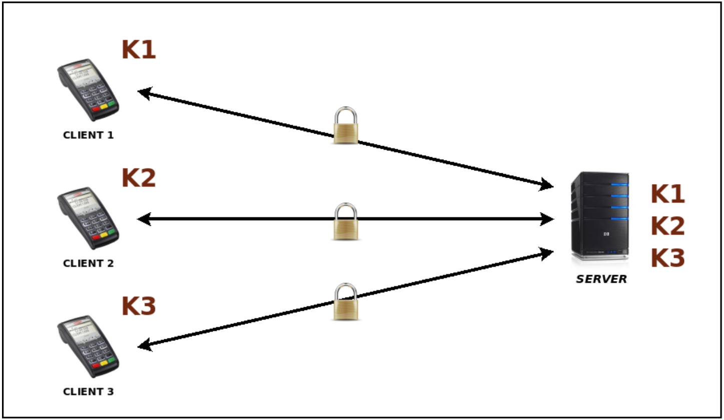
Clients would like to communicate with a server
(for example, PIN verification).

Motivation



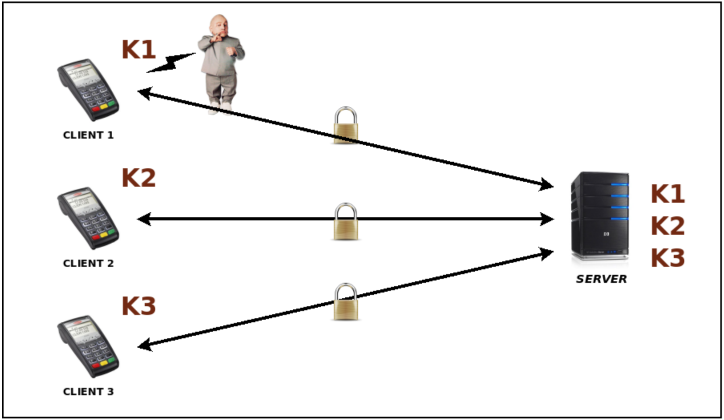
But attackers can eavesdrop on the communication channels.

Motivation



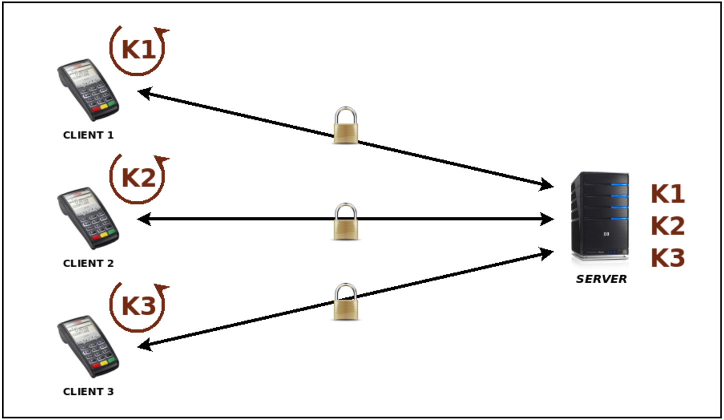
The channel can be protected using symmetric-key crypto (secret keys need to be shared during the initialization process and since this process is costly, the system should last as long as possible).

Motivation



We would like to cover the cases where the attacker could tamper with clients since they are located in unsecure areas.

Motivation



Thus, we would like the scheme to be forward secure on the client side (not in the server).

Deriving keys

In the following, in order to derive keys we consider that we only have access to a blackbox function F:

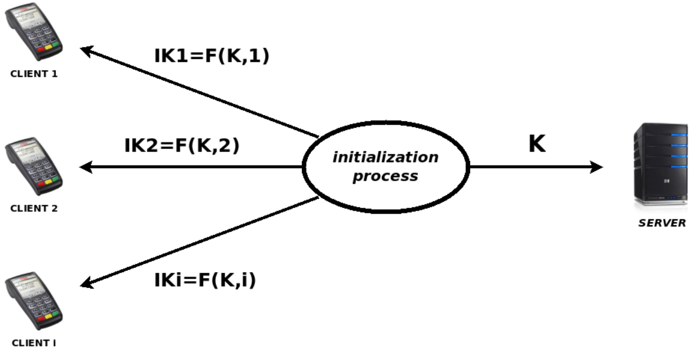
- that has two inputs: the original key $K \in \mathcal{K}$ and some arbitrary-length salt value s
- that outputs a new key $K' \in \mathcal{K}$

In practice, one can use the HKDF proposal (Krawczyk 2010) instantiated with HMAC or CBC-MAC, or directly:

$$K' = F(K, s) = trunc(HMAC(K, s))$$

Initialization of the system

We only have to study the problem reduced to one client:
we consider that an **initial key** IK_i is derived for each client i with $IK_i = F(K, i)$, while the server only stores K.



Parameters

We assume that the identity of the client and of the session key are publicly sent with the session protected messages.

Three parameters are important:

- R: the number of key registers available in the client's memory
- N: the maximal number of calls to F the server has to perform in order to retrieve one session key from the client initial key
- T: the maximal number of session key the system can handle after an initialization

Trivial cases

We can identify trivial cases:

- $R = 1$
  for each session, use the key stored in the client register and self-update it. We have $T = N$.

- $N = 1$
  during the initialization, fill all registers of the client with a different key. Then, for each session, use and erase a key in one of the R register. We have $T = R$.

More generally:

- initialize all R registers with a different key: $K_r = F(IK, r)$
- for a session j, we use the key located in register $r = j \pmod R$ and self-update this register with $K_r = F(K_r, j)$
- we thus have $T = N \times R$

Our results

In ANSI X9.24 a better solution is described:
the algorithm ***Derived Unique Key Per Transaction* (DUKPT)**, very much utilized in the banking industry.

- $R = 21, N = \lfloor R/2 \rfloor = 10$ and $T = 2^{R-1} - 1 = 1048575$
- not really scalable

We propose another algorithm, ***Optimal-DUKPT***:

- completely scalable
- very simple to understand/implement
- very good performances: $T = \binom{R+N}{N} - 1$
  ... actually optimal in R, N and T
- also better performances than DUKPT when N is the number of operations on average

Outline

Introduction and motivation

Derived Unique Key Per Transaction (DUKPT)

Optimal-DUKPT

Comparison and Optimality

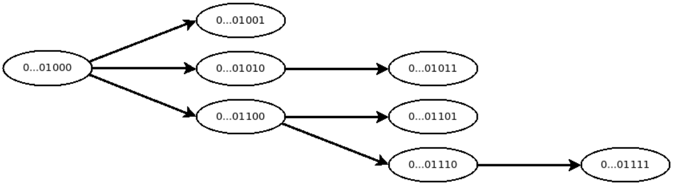
DUKPT: parameters and key hierarchy

Parameters:

- $T = 2^{20} - 1 \simeq 1000000$
- $R = 21$
- $N = 10$

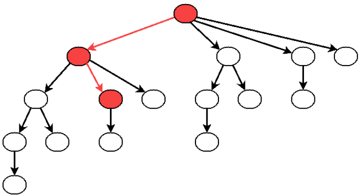
Key hierarchy:

- For $x \neq 0$, we define $y = \tilde{x}$ to be the value of x with the least significant "1" bit set to zero:
  if $x = (10110)_2$ we have $y = \tilde{x} = (10100)_2$ and $\tilde{y} = (10000)_2$.
- DUKPT intrinsically defines a hierarchy between the keys: each key used for session $j \neq 0$ is the daughter of the key identified by $\tilde{j}$: $K_j = F(K_{\tilde{j}}, j)$.
- we only use keys for the sessions j such that $HW(j) \leq 10$.



DUKPT: an example on the server side

For each session j:
the server deduces K_j by simply starting from the top node of the tree IK and recovers the successive keys during the path to K_j.



For example, if $j = (0\ldots011010)_2$, the server computes:

- $K_{0\ldots010000} = F(IK, (0\ldots010000)_2)$
- then $K_{0\ldots011000} = F(K_{0\ldots010000}, (0\ldots011000)_2)$
- and finally $K_j = F(K_{0\ldots011000}, (0\ldots011010)_2)$.

Note that we have $N \leq 10$ since $HW(j) \leq 10$.

DUKPT: an example on the client side

Initialization: all registers R_i are filled with $K_{2^{i-1}} = F(IK, 2^{i-1})$.

For each session j**:** the client picks and uses the key K_j located in register r, where r is the bit position of the least significant "1" bit of j. Then, before erasing K_j from its memory, the client derives and stores all the $r-1$ direct daughters of K_j in the $r-1$ least significant registers.

session j	R_{21}	R_{20}	R_{19}		R_{12}	R_{11}		R_5	R_4	R_3	R_2	R_1
init	2^{20}	2^{19}	2^{18}	...	2^{11}	2^{10}	...	16	8	4	2	1
1												X
2											X	3
3												X
4										X	6	5
5												X
6											X	7
7												X
8									X	12	10	9
9												X
10											X	11
11												X
12											14	13
13												X
14											X	15

Outline

Introduction and motivation

Derived Unique Key Per Transaction (DUKPT)

Optimal-DUKPT

Comparison and Optimality

Optimal-DUKPT: improving DUKPT

Idea to improve DUKPT:

- the implicit key hierarchy tree built by DUKPT is not optimal: many leaves are not a distance N
- instead, **Optimal-DUKPT will build a tree for which we are ensured that all leaves are at distance N**
- this will maximize the total number of nodes in the tree, thus maximize T

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				X
2			X	3
3				X
4		X	6	5
5				X
6			X	7
7				X
8	X	12	10	9
9				X
10			X	11
11				X

DUKPT (with $N = 3$)

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				X
2			X	3
3				X
4		X	6	5
5				X
6			X	7
7				X
8	X	12	10	9
9				X
10			X	11
11				X

Optimal-DUKPT (with $N = 3$)

Optimal-DUKPT: improving DUKPT

Idea to improve DUKPT:

- the implicit key hierarchy tree built by DUKPT is not optimal: many leaves are not a distance N
- instead, **Optimal-DUKPT will build a tree for which we are ensured that all leaves are at distance N**
- this will maximize the total number of nodes in the tree, thus maximize T

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				X
2			X	3
3				X
4		X	6	5
5				X
6			X	7
7				X
8	X	12	10	9
9				X
10			X	11
11				X

DUKPT (with $N = 3$)

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				1-a
2				X
3			X	3
4				X
5		X	6	5
6				X
7			X	7
8				X
9	X	12	10	9
10				X
11			X	11

Optimal-DUKPT (with $N = 3$)

Optimal-DUKPT: improving DUKPT

Idea to improve DUKPT:

- the implicit key hierarchy tree built by DUKPT is not optimal: many leaves are not a distance N
- instead, **Optimal-DUKPT will build a tree for which we are ensured that all leaves are at distance N**
- this will maximize the total number of nodes in the tree, thus maximize T

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				X
2			X	3
3				X
4		X	6	5
5				X
6			X	7
7				X
8	X	12	10	9
9				X
10			X	11
11				X

DUKPT (with $N = 3$)

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				1-a
2				1-b
3				X
4			X	3
5				X
6		X	6	5
7				X
8			X	7
9				X
10	X	12	10	9
11				X

Optimal-DUKPT (with $N = 3$)

Optimal-DUKPT: improving DUKPT

Idea to improve DUKPT:

- the implicit key hierarchy tree built by DUKPT is not optimal: many leaves are not a distance N
- instead, **Optimal-DUKPT will build a tree for which we are ensured that all leaves are at distance N**
- this will maximize the total number of nodes in the tree, thus maximize T

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				X
2			X	3
3				X
4		X	6	5
5				X
6			X	7
7				X
8	X	12	10	9
9				X
10			X	11
11				X

DUKPT (with $N = 3$)

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				1-a
2				1-b
3				X
4			2-a	3
5				X
6			X	
7		X	6	5
8				X
9			X	7
10				X
11	X	12	10	9

Optimal-DUKPT (with $N = 3$)

Optimal-DUKPT: improving DUKPT

Idea to improve DUKPT:

- the implicit key hierarchy tree built by DUKPT is not optimal: many leaves are not a distance N
- instead, **Optimal-DUKPT will build a tree for which we are ensured that all leaves are at distance N**
- this will maximize the total number of nodes in the tree, thus maximize T

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				X
2			X	3
3				X
4		X	6	5
5				X
6			X	7
7				X
8	X	12	10	9
9				X
10			X	11
11				X

DUKPT (with $N = 3$)

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				1-a
2				1-b
3				X
4			2-a	3
5				3-a
6				X
7			X	
8		X	6	5
9				X
10			X	7
11				X

Optimal-DUKPT (with $N = 3$)

Optimal-DUKPT: improving DUKPT

Idea to improve DUKPT:

- the implicit key hierarchy tree built by DUKPT is not optimal: many leaves are not a distance N
- instead, **Optimal-DUKPT will build a tree for which we are ensured that all leaves are at distance N**
- this will maximize the total number of nodes in the tree, thus maximize T

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				X
2			X	3
3				X
4		X	6	5
5				X
6			X	7
7				X
8	X	12	10	9
9				X
10			X	11
11				X

DUKPT (with $N = 3$)

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				1-a
2				1-b
3				X
4			2-a	3
5				3-a
6				X
7			2-b	3-b
8				X
9			X	
10		X	6	5
11				X

Optimal-DUKPT (with $N = 3$)

Optimal-DUKPT: improving DUKPT

Idea to improve DUKPT:

- the implicit key hierarchy tree built by DUKPT is not optimal: many leaves are not a distance N
- instead, **Optimal-DUKPT will build a tree for which we are ensured that all leaves are at distance N**
- this will maximize the total number of nodes in the tree, thus maximize T

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				X
2			X	3
3				X
4		X	6	5
5				X
6			X	7
7				X
8	X	12	10	9
9				X
10			X	11
11				X

DUKPT (with $N = 3$)

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				1-a
2				1-b
3				X
4			2-a	3
5				3-a
6				X
7			2-b	3-b
8				X
9			X	
10		4-a	6	5
11				X

Optimal-DUKPT (with $N = 3$)

Optimal-DUKPT: improving DUKPT

Idea to improve DUKPT:

- the implicit key hierarchy tree built by DUKPT is not optimal: many leaves are not a distance N
- instead, **Optimal-DUKPT will build a tree for which we are ensured that all leaves are at distance N**
- this will maximize the total number of nodes in the tree, thus maximize T

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				X
2			X	3
3				X
4		X	6	5
5				X
6			X	7
7				X
8	X	12	10	9
9				X
10			X	11
11				X

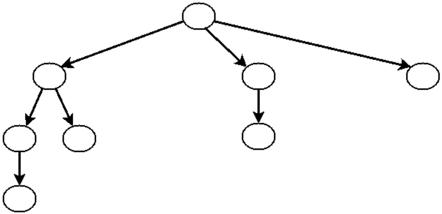
DUKPT (with $N = 3$)

session j	R_4	R_3	R_2	R_1
init	8	4	2	1
1				1-a
2				1-b
3				X
4			2-a	3
5				3-a
6				X
7			2-b	3-b
8				X
9			X	
10		4-a	6	5
11				5-a

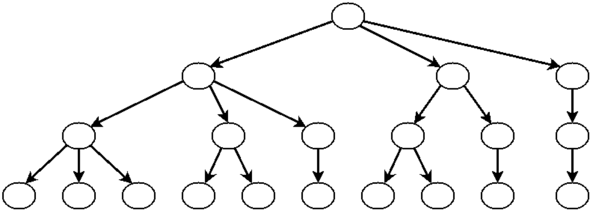
Optimal-DUKPT (with $N = 3$)

Tree comparison with $R = 3, N = 3$

For DUKPT, $T = 7$**:**



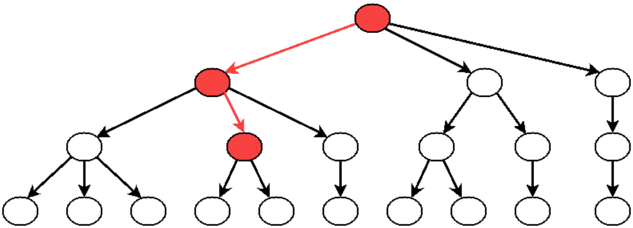
For Optimal-DUKPT, $T = 19$**:**



Optimal-DUKPT: an example on the server side

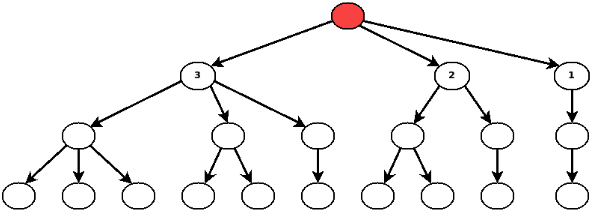
For each session *j*:

as for DUKPT, from the key identity j sent by the client, the server deduces K_j by simply starting from the top node of the tree IK and recovers the successive keys during the path to K_j.



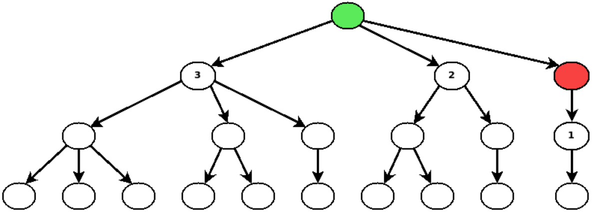
Optimal-DUKPT: an example on the client side (with $R = 3$ and $N = 3$)

session j	R_3	R_2	R_1	distance		
				$d(R_3)$	$d(R_2)$	$d(R_1)$
init	K_{10}	K_4	K_1	1	1	1
1			K_2	1	1	2
2			K_3	1	1	3
3			X	1	1	X
4		K_7	K_5	1	2	2
5			K_6	1	2	3
6			X	1	2	X
7		K_9	K_8	1	3	3
8			X	1	3	X
9		X		1	X	X
10	K_{16}	K_{13}	K_{11}	2	2	2



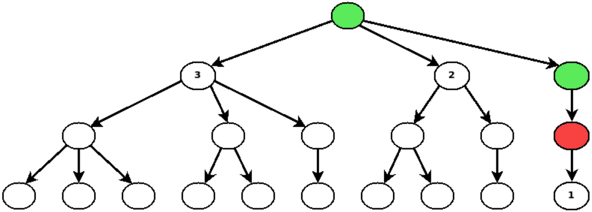
Optimal-DUKPT: an example on the client side (with $R = 3$ and $N = 3$)

session j	R_3	R_2	R_1	distance		
				$d(R_3)$	$d(R_2)$	$d(R_1)$
init	K_{10}	K_4	K_1	1	1	1
1			K_2	1	1	2
2			K_3	1	1	3
3			X	1	1	X
4		K_7	K_5	1	2	2
5			K_6	1	2	3
6			X	1	2	X
7		K_9	K_8	1	3	3
8			X	1	3	X
9		X		1	X	X
10	K_{16}	K_{13}	K_{11}	2	2	2



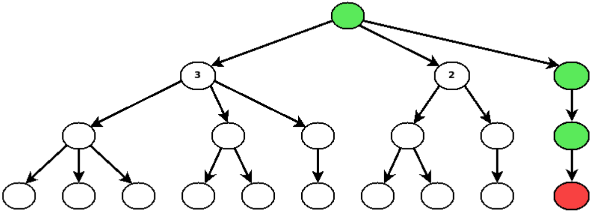
Optimal-DUKPT: an example on the client side (with $R = 3$ and $N = 3$)

session j	R_3	R_2	R_1	distance		
				$d(R_3)$	$d(R_2)$	$d(R_1)$
init	K_{10}	K_4	K_1	1	1	1
1			K_2	1	1	2
2			K_3	1	1	3
3			X	1	1	X
4		K_7	K_5	1	2	2
5			K_6	1	2	3
6			X	1	2	X
7		K_9	K_8	1	3	3
8			X	1	3	X
9		X		1	X	X
10	K_{16}	K_{13}	K_{11}	2	2	2



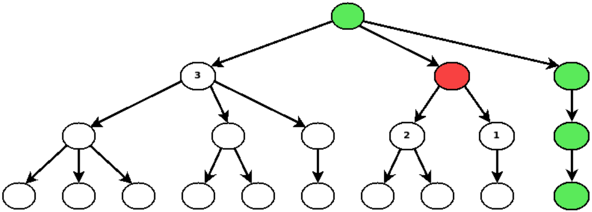
Optimal-DUKPT: an example on the client side (with $R = 3$ and $N = 3$)

session j	R_3	R_2	R_1	distance		
				$d(R_3)$	$d(R_2)$	$d(R_1)$
init	K_{10}	K_4	K_1	1	1	1
1			K_2	1	1	2
2			K_3	1	1	3
3			X	1	1	X
4		K_7	K_5	1	2	2
5			K_6	1	2	3
6			X	1	2	X
7		K_9	K_8	1	3	3
8			X	1	3	X
9		X		1	X	X
10	K_{16}	K_{13}	K_{11}	2	2	2



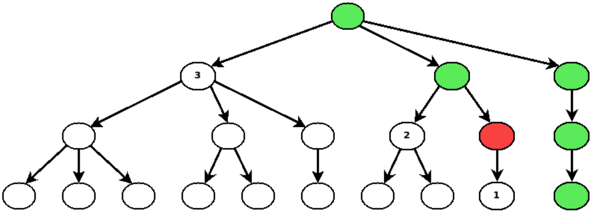
Optimal-DUKPT: an example on the client side (with $R = 3$ and $N = 3$)

session j	R_3	R_2	R_1	distance		
				$d(R_3)$	$d(R_2)$	$d(R_1)$
init	K_{10}	K_4	K_1	1	1	1
1			K_2	1	1	2
2			K_3	1	1	3
3			X	1	1	X
4		K_7	K_5	1	2	2
5			K_6	1	2	3
6			X	1	2	X
7		K_9	K_8	1	3	3
8			X	1	3	X
9		X		1	X	X
10	K_{16}	K_{13}	K_{11}	2	2	2



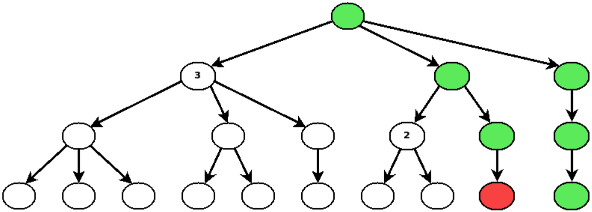
Optimal-DUKPT: an example on the client side (with $R = 3$ and $N = 3$)

session j	R_3	R_2	R_1	distance		
				$d(R_3)$	$d(R_2)$	$d(R_1)$
init	K_{10}	K_4	K_1	1	1	1
1			K_2	1	1	2
2			K_3	1	1	3
3			X	1	1	X
4		K_7	K_5	1	2	2
5			K_6	1	2	3
6			X	1	2	X
7		K_9	K_8	1	3	3
8			X	1	3	X
9		X		1	X	X
10	K_{16}	K_{13}	K_{11}	2	2	2



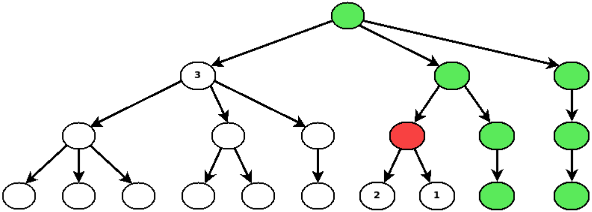
Optimal-DUKPT: an example on the client side (with $R = 3$ and $N = 3$)

session j	R_3	R_2	R_1	distance		
				$d(R_3)$	$d(R_2)$	$d(R_1)$
init	K_{10}	K_4	K_1	1	1	1
1			K_2	1	1	2
2			K_3	1	1	3
3			X	1	1	X
4		K_7	K_5	1	2	2
5			K_6	1	2	3
6			X	1	2	X
7		K_9	K_8	1	3	3
8			X	1	3	X
9		X		1	X	X
10	K_{16}	K_{13}	K_{11}	2	2	2



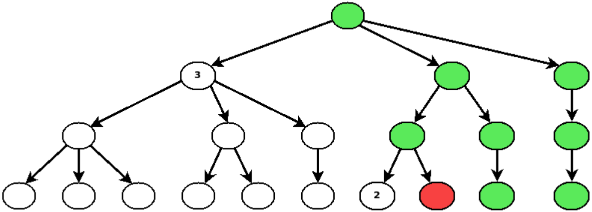
Optimal-DUKPT: an example on the client side (with $R = 3$ and $N = 3$)

session j	R_3	R_2	R_1	distance		
				$d(R_3)$	$d(R_2)$	$d(R_1)$
init	K_{10}	K_4	K_1	1	1	1
1			K_2	1	1	2
2			K_3	1	1	3
3			X	1	1	X
4		K_7	K_5	1	2	2
5			K_6	1	2	3
6			X	1	2	X
7		K_9	K_8	1	3	3
8			X	1	3	X
9		X		1	X	X
10	K_{16}	K_{13}	K_{11}	2	2	2



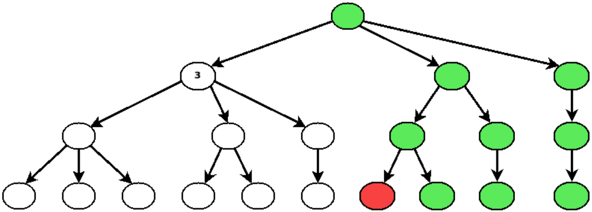
Optimal-DUKPT: an example on the client side (with $R = 3$ and $N = 3$)

session j	R_3	R_2	R_1	distance		
				$d(R_3)$	$d(R_2)$	$d(R_1)$
init	K_{10}	K_4	K_1	1	1	1
1			K_2	1	1	2
2			K_3	1	1	3
3			X	1	1	X
4		K_7	K_5	1	2	2
5			K_6	1	2	3
6			X	1	2	X
7		K_9	K_8	1	3	3
8			X	1	3	X
9		X		1	X	X
10	K_{16}	K_{13}	K_{11}	2	2	2



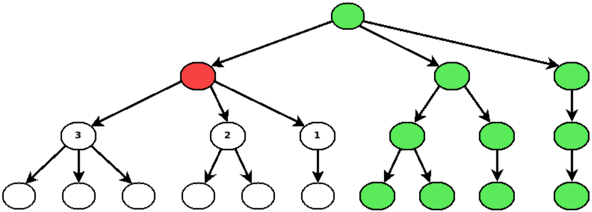
Optimal-DUKPT: an example on the client side (with $R = 3$ and $N = 3$)

session j	R_3	R_2	R_1	distance		
				$d(R_3)$	$d(R_2)$	$d(R_1)$
init	K_{10}	K_4	K_1	1	1	1
1			K_2	1	1	2
2			K_3	1	1	3
3			X	1	1	X
4		K_7	K_5	1	2	2
5			K_6	1	2	3
6			X	1	2	X
7		K_9	K_8	1	3	3
8			X	1	3	X
9		X		1	X	X
10	K_{16}	K_{13}	K_{11}	2	2	2



Optimal-DUKPT: an example on the client side (with $R = 3$ and $N = 3$)

session j	R_3	R_2	R_1	distance		
				$d(R_3)$	$d(R_2)$	$d(R_1)$
init	K_{10}	K_4	K_1	1	1	1
1			K_2	1	1	2
2			K_3	1	1	3
3			X	1	1	X
4		K_7	K_5	1	2	2
5			K_6	1	2	3
6			X	1	2	X
7		K_9	K_8	1	3	3
8			X	1	3	X
9		X		1	X	X
10	K_{16}	K_{13}	K_{11}	2	2	2



Outline

Introduction and motivation

Derived Unique Key Per Transaction (DUKPT)

Optimal-DUKPT

Comparison and Optimality

Comparison DUKPT / Optimal-DUKPT

	DUKPT ($R = 21, N = 10$)	O-DUKPT ($R = 21, N = 7$)	O-DUKPT ($R = 13, N = 10$)	O-DUKPT ($R = 17, N = 8$)
T	1048575	1184039	1144065	1081574
$A(1)/T$	$2^{-15.6}$	$2^{-15.8}$	$2^{-16.4}$	$2^{-16.0}$
$A(2)/T$	$2^{-12.3}$	$2^{-12.3}$	$2^{-13.6}$	$2^{-12.8}$
$A(3)/T$	$2^{-9.6}$	$2^{-9.4}$	$2^{-11.3}$	$2^{-10.1}$
$A(4)/T$	$2^{-7.4}$	$2^{-6.8}$	$2^{-9.3}$	$2^{-7.8}$
$A(5)/T$	$2^{-5.7}$	$2^{-4.5}$	$2^{-7.5}$	$2^{-5.7}$
$A(6)/T$	$2^{-4.3}$	$2^{-2.4}$	$2^{-5.9}$	$2^{-3.9}$
$A(7)/T$	$2^{-3.2}$	$2^{-0.4}$	$2^{-4.5}$	$2^{-2.1}$
$A(8)/T$	$2^{-2.4}$		$2^{-3.2}$	$2^{-0.6}$
$A(9)/T$	$2^{-1.8}$		$2^{-2.0}$	
$A(10)/T$	$2^{-1.6}$		$2^{-0.8}$	
C_S	8.65	6.68	9.28	7.56

$A(i)$ represents the number of keys at distance i

C_S stands for the average number of computations required to derive one key on the server side

Optimality

Let $\mathcal{A}$ be an optimal algorithm, i.e. reaching the maximum value T of keys handled. Sketch of the optimality proof:

Lemma 1

After the initialization process of $\mathcal{A}$, the R registers of the client are filled with R new distinct keys.

Lemma 2

When $\mathcal{A}$ derives keys on the client side during the registers update, it only memorizes newly derived keys in empty registers.

Lemma 3

When $\mathcal{A}$ derives keys on the client side during the registers update, all previously empty registers are filled at the end of the process.

Lemma 4

The transaction key chosen by $\mathcal{A}$ is always one of the keys at the maximal available distance from IK (different from $N + 1$).