

# Diffie-Hellman in CryptoVerif

Bruno Blanchet

CNRS, École Normale Supérieure, INRIA, Paris

June 2009

# Outline

- 1 Decisional Diffie-Hellman (DDH) assumption, basic model.
- 2 Computational Diffie-Hellman (CDH) assumption, basic model.
- 3 Why this is not enough for protocols relying on Diffie-Hellman key agreements.
- 4 Computational Diffie-Hellman (CDH) assumption, extended model.

# Decisional Diffie-Hellman assumption

Consider a multiplicative cyclic group  $G$  of order  $q$ , with generator  $g$ . A probabilistic polynomial-time adversary has a negligible probability of distinguishing

$$(g^a, g^b, g^{ab}) \text{ for random } a, b \in \mathbb{Z}_q$$

and

$$(g^a, g^b, g^c) \text{ for random } a, b, c \in \mathbb{Z}_q$$

# Decisional Diffie-Hellman assumption in CryptoVerif

Consider a multiplicative cyclic group  $G$  of order  $q$ , with generator  $g$ .  
 A probabilistic polynomial-time adversary has a negligible probability of distinguishing

$$(g^a, g^b, g^{ab}) \text{ for random } a, b \in \mathbb{Z}_q$$

and

$$(g^a, g^b, g^c) \text{ for random } a, b, c \in \mathbb{Z}_q$$

In CryptoVerif, this can be written

```
!i≤N new a : Z; new b : Z;
  (( () → exp(g, a), () → exp(g, b), () → exp(g, mult(a, b)))
```

≈

```
!i≤N new a : Z; new b : Z; new c : Z;
  (( () → exp(g, a), () → exp(g, b), () → exp(g, c))
```

# Decisional Diffie-Hellman assumption in CryptoVerif

$g : G$	generator of $G$
$\text{exp}(G, Z) : G$	exponentiation
$\text{mult}(Z, Z) : Z$ commutative	product in $\mathbb{Z}_q$
$\text{exp}(\text{exp}(z, a), b) = \text{exp}(z, \text{mult}(a, b))$	$(z^a)^b = z^{ab}$

$(g^a)^b = g^{ab}$  and  $(g^b)^a = g^{ba}$ , equal by commutativity of *mult*

$!^{i \leq N}$  **new**  $a : Z$ ; **new**  $b : Z$ ;  
 $((\ ) \rightarrow \text{exp}(g, a), (\ ) \rightarrow \text{exp}(g, b), (\ ) \rightarrow \text{exp}(g, \text{mult}(a, b)))$

$\approx$

$!^{i \leq N}$  **new**  $a : Z$ ; **new**  $b : Z$ ; **new**  $c : Z$ ;  
 $((\ ) \rightarrow \text{exp}(g, a), (\ ) \rightarrow \text{exp}(g, b), (\ ) \rightarrow \text{exp}(g, c))$

We replace  $g^{ab}$  with  $g^c$  for some fresh random number  $c$ , provided  $a$  and  $b$  are random numbers used only in  $g^a$ ,  $g^b$ , and  $g^{ab}$ .

# Application: semantic security of El Gamal (A. Chaudhuri)

```
start();  
new  $x : Z$ ;  
let  $alpha : G = exp(g, x)$  in  
 $\overline{c_{PK}}\langle alpha \rangle$ ;  
 $c_E(m_0 : G, m_1 : G)$ ;  
new  $b : bool$ ;  
let  $m : G = choose(b, m_0, m_1)$  in  
new  $y : Z$ ;  
let  $beta : G = exp(g, y)$  in  
let  $delta : G = exp(alpha, y)$  in  
let  $zeta : G = dot(delta, m)$  in  
 $\overline{c_{Eret}}\langle beta, zeta \rangle$ 
```

# Application: semantic security of El Gamal (A. Chaudhuri)

```
start();  
new  $x : Z$ ;  
let  $alpha : G = exp(g, x)$  in  
 $\overline{c_{PK}}\langle alpha \rangle$ ;  
 $c_E(m_0 : G, m_1 : G)$ ;  
new  $b : bool$ ;  
let  $m : G = choose(b, m_0, m_1)$  in  
new  $y : Z$ ;  
let  $beta : G = exp(g, y)$  in  
let  $delta : G = exp(g, mult(x, y))$  in  
let  $zeta : G = dot(delta, m)$  in  
 $\overline{c_{Eret}}\langle beta, zeta \rangle$ 
```

# Application: semantic security of El Gamal (A. Chaudhuri)

```
start();  
new z : Z;  
new x : Z;  
let alpha : G = exp(g, x) in  
 $\overline{c_{PK}}$ (alpha);  
cE(m0 : G, m1 : G);  
new b : bool;  
let m : G = choose(b, m0, m1) in  
new y : Z;  
let beta : G = exp(g, y) in  
let delta : G = exp(g, z) in  
let zeta : G = dot(delta, m) in  
 $\overline{c_{Ret}}$ (beta, zeta)
```

by DDH.

# Application: semantic security of El Gamal (A. Chaudhuri)

```

start();
new alpha : G;
 $\overline{c_{PK}}$ (alpha);
 $c_E(m_0 : G, m_1 : G)$ ;
new b : bool;
let m : G = choose(b, m_0, m_1) in
new beta : G;
new delta : G;
let zeta : G = dot(delta, m) in
 $\overline{c_{Eret}}$ (beta, zeta)

by  $!^{i \leq n}$  new x : Z; ()  $\rightarrow$  exp(g, x)  $\approx$   $!^{i \leq n}$  new y : G; ()  $\rightarrow$  y.

```

# Application: semantic security of El Gamal (A. Chaudhuri)

```

start();
new alpha : G;
 $\overline{c_{PK}}$  $\langle$ alpha $\rangle$ ;
 $c_E(m_0 : G, m_1 : G)$ ;
new b : bool;
let m : G = choose(b, m_0, m_1) in
new beta : G;
new zeta : G;
 $\overline{c_{Eret}}$  $\langle$ beta, zeta $\rangle$ 

by  $!^{i \leq n}$  new x : G; (y : G)  $\rightarrow$  dot(x, y)  $\approx$   $!^{i \leq n}$  new x : G; (y : G)  $\rightarrow$  x.

```

# Application: semantic security of El Gamal (A. Chaudhuri)

```
start();  
new alpha : G;  
 $\overline{c_{PK}}$ (alpha);  
cE(m0 : G, m1 : G);  
new b : bool;  
new beta : G;  
new zeta : G;  
 $\overline{c_{Eret}}$ (beta, zeta)
```

**b** is **secret**, which proves semantic security of El Gamal.

# Computational Diffie-Hellman assumption

Consider a multiplicative cyclic group  $G$  of order  $q$ , with generator  $g$ . A probabilistic polynomial-time adversary has a negligible probability of **computing**  $g^{ab}$  from  $g$ ,  $g^a$ ,  $g^b$ , for random  $a, b \in \mathbb{Z}_q$ .

# Computational Diffie-Hellman assumption in CryptoVerif

Consider a multiplicative cyclic group  $G$  of order  $q$ , with generator  $g$ . A probabilistic polynomial-time adversary has a negligible probability of **computing**  $g^{ab}$  from  $g$ ,  $g^a$ ,  $g^b$ , for random  $a, b \in \mathbb{Z}_q$ .

In CryptoVerif, this can be written

$!^{i \leq N}$  **new**  $a : Z$ ; **new**  $b : Z$ ;

$((() \rightarrow \text{exp}(g, a), () \rightarrow \text{exp}(g, b), !^{i' \leq N'} (z : G) \rightarrow z = \text{exp}(g, \text{mult}(a, b))))$

$\approx$

$!^{i \leq N}$  **new**  $a : Z$ ; **new**  $b : Z$ ; **new**  $c : Z$ ;

$((() \rightarrow \text{exp}(g, a), () \rightarrow \text{exp}(g, b), !^{i' \leq N'} (z : G) \rightarrow \text{false}))$

# Computational Diffie-Hellman assumption in CryptoVerif

Consider a multiplicative cyclic group  $G$  of order  $q$ , with generator  $g$ . A probabilistic polynomial-time adversary has a negligible probability of **computing**  $g^{ab}$  from  $g$ ,  $g^a$ ,  $g^b$ , for random  $a, b \in \mathbb{Z}_q$ .

In CryptoVerif, this can be written

$!^{i \leq N}$  **new**  $a : Z$ ; **new**  $b : Z$ ;

$((\ () \rightarrow \text{exp}(g, a), (\ () \rightarrow \text{exp}(g, b), !^{i' \leq N'} (z : G) \rightarrow z = \text{exp}(g, \text{mult}(a, b))))$

$\approx$

$!^{i \leq N}$  **new**  $a : Z$ ; **new**  $b : Z$ ; **new**  $c : Z$ ;

$((\ () \rightarrow \text{exp}(g, a), (\ () \rightarrow \text{exp}(g, b), !^{i' \leq N'} (z : G) \rightarrow \text{false}))$

(Could be extended by allowing to reveal  $g^{ab}$  and replacing  $z = g^{ab}$  with *false* only when  $g^{ab}$  has not been revealed. Similar to one-wayness.)

# Computational Diffie-Hellman assumption

Consider a multiplicative cyclic group  $G$  of order  $q$ , with generator  $g$ . A probabilistic polynomial-time adversary has a negligible probability of **computing**  $g^{ab}$  from  $g$ ,  $g^a$ ,  $g^b$ , for random  $a, b \in \mathbb{Z}_q$ .

In CryptoVerif, this can be written

$!^{i \leq N}$  **new**  $a : Z$ ; **new**  $b : Z$ ;

$((\ () \rightarrow \text{exp}(g, a), (\ () \rightarrow \text{exp}(g, b), !^{i' \leq N'} (z : G) \rightarrow z = \text{exp}(g, \text{mult}(a, b))))$

$\approx$

$!^{i \leq N}$  **new**  $a : Z$ ; **new**  $b : Z$ ; **new**  $c : Z$ ;

$((\ () \rightarrow \text{exp}(g, a), (\ () \rightarrow \text{exp}(g, b), !^{i' \leq N'} (z : G) \rightarrow \text{false}))$

(Could be extended by allowing to reveal  $g^{ab}$  and replacing  $z = g^{ab}$  with *false* only when  $g^{ab}$  has not been revealed. Similar to one-wayness.)

Application: semantic security of **hashed El Gamal in the random oracle model** (A. Chaudhuri).

# Typical protocol using the Diffie-Hellman key agreement

Assumptions on primitives:

- $h$  hash function in the random oracle model
- CDH assumption

A simplified form of a Diffie-Hellman key agreement protocol:

Message 1.  $A \rightarrow B$ :  $g^a$  for random  $a$

Message 2.  $B \rightarrow A$ :  $g^b$  for random  $b$

The shared key is  $h(g^{ab}) = h((g^a)^b) = h((g^b)^a)$

(Signatures omitted for simplicity.)

# Typical protocol using the Diffie-Hellman key agreement in CryptoVerif

```

!iA≤N cA(); new a : Z; cA⟨exp(g, a)⟩; cA(gb).let k = h(exp(gb, a)) in ...
|
!iB≤N cB(); new b : Z; cB⟨exp(g, b)⟩; cA(ga).let k = h(exp(ga, b)) in ...
|
!iH≤nH cH(x); cH⟨h(x)⟩

```

Cannot be transformed by the previous CDH equivalence, because  $a$  and  $b$  are chosen in parallel processes, not one after the other under the same replication.

# Extending the formalization of CDH in CryptoVerif

After applying the security assumption on the hash function  $h$ ,

- $h(x)$  returns a fresh random number if  $h(x)$  has not already been called,
- and the same result as the previous call otherwise.

Hence  $h(x)$  is replaced with lookups that **compare  $x$  with the other arguments of  $h$** .

$$!^{iA \leq N} cA(); \text{new } a : Z; \overline{cA} \langle \exp(g, a) \rangle; cA(gb) \dots \exp(gb[u], a[u]) = \exp(gb, a) \\ \dots \exp(ga[u'], b[u']) = \exp(gb, a) \dots x[u''] = \exp(gb, a) \dots$$

|

$$!^{iB \leq N} cB(); \text{new } b : Z; \overline{cB} \langle \exp(g, b) \rangle; cA(ga) \dots \exp(gb[u], a[u]) = \exp(ga, b) \\ \dots \exp(ga[u'], b[u']) = \exp(ga, b) \dots x[u''] = \exp(ga, b) \dots$$

|

$$!^{iH \leq nH} cH(x); \dots \exp(gb[u], a[u]) = x \dots \exp(ga[u'], b[u']) = x \dots x[u''] = x.$$

# Extending the formalization of CDH in CryptoVerif

$$\begin{aligned}
 & !^{i \leq n} \text{ new } a : Z; ( () \rightarrow \text{exp}(g, a), \\
 & \quad !^{i_2 \leq n_2} (m : G, r : G) \rightarrow r = \text{exp}(m, a), \\
 & \quad !^{i_3 \leq n_3} (m : G, m' : G, i' \leq n) \rightarrow \text{exp}(m, a) = \text{exp}(m', a[i']) ) \\
 & \approx \\
 & !^{i \leq n} \text{ new } a : Z; ( () \rightarrow \text{exp}(g, a), \\
 & \quad !^{i_2 \leq n_2} (m : G, r : G) \rightarrow \text{find } i' \leq n \text{ suchthat defined}(a[i']) \wedge \\
 & \quad \quad m = \text{exp}(g, a[i']) \text{ then false else } r = \text{exp}(m, a), \\
 & \quad !^{i_3 \leq n_3} (m : G, m' : G, i' \leq n) \rightarrow \text{exp}(m, a) = \text{exp}(m', a[i']) )
 \end{aligned}$$

Using **an array index  $i'$  as argument** of a function is new with respect what was used for previous primitives.

The implementation of this extension is in progress.