Publicly Verifiable Secret Sharing for Cloud-based Key Management

Roy D'Souza, David Jao, Ilya Mironov and Omkant Pandey

Microsoft Corporation and University of Waterloo

December 13, 2011

Overview

Motivation:

- Allow users to store encrypted files in untrusted cloud servers.
- Experience shows that some proportion of users will forget their keys, necessitating *key recovery* services.
- One way to perform key recovery is via trusted third parties.

Results:

- We define the notion of a public-key encryption scheme supporting publicly-verifiable secret sharing.
- We construct a PKE-supporting-PVSS scheme secure under DBDH.
- Our scheme is also the first (plain) PVSS scheme provably secure in the standard model.

Access structures

Let $\{P_1, \ldots, P_n\}$ be a set of parties.

- A collection $\mathbb{A} \subseteq 2^{\{P_1,\ldots,P_n\}}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$.
- An *access structure* (resp., monotone access structure) is a collection (resp., monotone collection) $\mathbb{A}$ of non-empty subsets of $\{P_1, \ldots, P_n\}$.
- The sets in $\mathbb{A}$ are called *authorized sets*, and the sets not in $\mathbb{A}$ are called *unauthorized sets*.

In this work we consider access structures (necessarily monotone) that are representable by a tree of threshold gates.

Public-key encryption scheme supporting Publicly-Verifiable Secret Sharing

A PKE supporting PVSS for an access structure $\mathbb{A}$ consists of algorithms $\{\mathcal{K}, \mathcal{E}, \mathcal{D}, \texttt{Setup}, \texttt{GenShare}, \texttt{Verify}, \texttt{Reconst}\}$ where $\mathrm{PKE} = \{\mathcal{K}, \mathcal{E}, \mathcal{D}\}$ is a public-key encryption scheme and:

- $\texttt{Setup}(1^\kappa, n) : i \in [1, n] \mapsto \{(PP_1, SK_1), \ldots, (PP_n, SK_n)\}$
- $\texttt{GenShare}(PK, SK, \mathbb{A}) : (PK, SK, \mathbb{A}) \mapsto \pi$
- $\texttt{Verify}(PK, \pi, \mathbb{A})$: outputs either 1 or 0 where

$$\mathrm{Prob}[\texttt{Verify}(PK, \pi, \mathbb{A}) = 1 : (PK, SK) \leftarrow \mathcal{K}(1^\kappa) \wedge \pi \leftarrow \texttt{GenShare}(PK, SK, \mathbb{A})] = 1.$$

- $\texttt{Reconst}(PK, \pi, \mathbb{A}, SK_S)$: reconstructs the secret key SK from π, where $S \in \mathbb{A}$ is an authorized set.

Related work

- Stadler, Eurocrypt 1996: First PVSS scheme. Can easily be adapted to support public-key encryption.
- Schoenmakers, Crypto 1999: Fastest extant PVSS scheme. Does not support public-key encryption.

The scheme: Key generation, encryption, and decryption

Let $e\colon \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be a pairing.

Key Generation $\mathcal{K}$**:** $h \xleftarrow{\$} \mathbb{G}_1$. $SK = h$, and $PK = e(g, h)$.

Encryption $\mathcal{E}_{PK}(m \in \mathbb{G}_2)$: $R \xleftarrow{\$} \mathbb{Z}_p$, output: $\langle g^R, m \cdot PK^R \rangle$.

Decryption $\mathcal{D}(\langle C_1, C_2 \rangle, SK)$: Output $C_2 / e(C_1, SK)$.

The scheme: Setup and share generation

- $\texttt{Setup}(1^\kappa, n)$: For every $i \in [1, n]$: sample $y_i \xleftarrow{\$} \mathbb{Z}_p$; output $SK_i = y_i$ and $PP_i = g^{y_i}$.
- $\texttt{GenShare}(PK, SK, \mathcal{T})$: Choose a polynomial q_x for every node x (including the leaves) in the $\mathcal{T}$.
  - For the root node r, set $q_r(0) = s$. Choose d_r more points randomly to completely fix the polynomial q_r.
  - For every other node x, set $q_x(0) = q_{\text{parent}(x)}(\text{id}(x))$; i.e., the constant term of q_x is set to $q_{\text{parent}(x)}(\text{id}(x))$. Choose the remaining d_x points randomly to completely define the polynomial q_x.
  - *Encapsulate shares*: For every *leaf* node x, the share of node x is defined by: $\lambda_x = g^{q_x(\text{id}(x))}$ (compute using polynomial interpolation).
  - For every node x and every $0 \leq i \leq d_x$, define the following values: $A_{x,i} = g^{q_x(i)} \quad \text{and} \quad \widehat{A}_{x,i} = e(g, A_{x,i}) = e(g, g)^{q_x(i)}$.

The scheme: Share generation

The output string π consists of the following:

1. For every node x (including the leaf nodes), the "committed polynomial": $\{\widehat{A}_{x,i}\}_{i=1}^{d_x}$;
2. For every leaf node, the encapsulations: $\langle B_x, C_x \rangle$.

The scheme: Verification

$\texttt{Verify}(PK, \pi, \mathcal{T})$:

1. For every node x in $\mathcal{T}$, parse π to obtain the committed points $\{\widehat{A}_{x,i}\}_{i=1}^{d_x}$ of polynomial q_x. For every leaf node x in $\mathcal{T}$, parse π to obtain the encapsulations $\langle B_x, C_x \rangle$ of secrets λ_x.
2. For the root node, verify that $\widehat{A}_{r,0} = PK$. For every other node x, verify that:

$$\widehat{A}_{x,0} = \prod_{i=0}^{d_z} \left(\widehat{A}_{z,i}\right)^{\Delta_{i,\gamma_z}(w)}, \tag{1}$$

   where $z = \mathrm{parent}(x)$, $w = \mathrm{id}(x)$, and $\gamma_z = \{0, 1, \ldots, d_z\}$.

3. For every leaf node x, verify that:

$$\widehat{A}_{x,0} = \frac{e(g, C_x)}{e(B_x, PP_i)}, \tag{2}$$

   where $i = \mathrm{id}(x)$.

If all tests pass, output 1; otherwise output 0.

The scheme: Reconstruction

To define $\texttt{Reconst}(PK, \pi, \mathcal{T}, SK_S)$ we define a recursive algorithm $\mathrm{DecryptNode}(\pi, SK_S, x)$ that outputs an element in $\mathbb{G}_1$ or $\perp$.

- If x is a leaf node then let $y_i \in SK_S$ be the secret key corresponding to PP_i where $i = \mathrm{id}(x)$. Set

$$\mathrm{DecryptNode}(\pi, SK_S, x) = \frac{C_x}{B_x^{y_i}} = \frac{\lambda_x \cdot PP_i^{R_x}}{g^{R_x \cdot y_i}} = \lambda_x = g^{q_x(0)}$$

for $i \in S$ and $\mathrm{DecryptNode}(\pi, SK_S, x) = \perp$ for $i \notin S$.

The scheme: Reconstruction

If x is not a leaf node:

- For all nodes z that are *children* of x, call $\mathrm{DecryptNode}(\pi, SK_S, z)$ and store the output as F_z.
- Let γ_x be an arbitrary k_x-sized set of child nodes z such that $F_z \neq \perp$. (If no such set exists then return $\perp$.)
- Compute:

$$F_x = \prod_{z \in \gamma_x} F_z^{\Delta_{i,\gamma_x'}(0)} = \prod_{z \in \gamma_x} g^{q_z(0) \cdot \Delta_{i,\gamma_x'}(0)}$$

$$= \prod_{z \in \gamma_x} g^{q_{\mathrm{parent}(z)}(\mathrm{id}(z)) \cdot \Delta_{i,\gamma_x'}(0)} = \prod_{z \in \gamma_x} g^{q_x(i) \cdot \Delta_{i,\gamma_x'}(0)} = g^{q_x(0)}$$

  where $i = \mathrm{id}(z)$ and $\gamma_x' = \{\mathrm{id}(z) : z \in \gamma_x\}$.
- Set $\mathtt{Reconst}(PK, \pi, \mathcal{T}, SK_S) = \mathrm{DecryptNode}(\pi, SK_S, r)$.

Security for our PKE supporting PVSS scheme

Theorem

If a polynomial time adversary $\mathcal{A}$ wins the security game for PKE scheme supporting publicly verifiable secret-sharing scheme, then there exists a polynomial time simulator $\mathcal{B}$ to break the Bilinear Diffie-Hellman Assumption.

See paper for the definition of the PKE-supporting-PVSS security game and the proof of the theorem.

Performance: Share generation

128 bit	$k = 1$	5	10	15	20	25	30	35	40	45	50
$n = 10$	760 830	760 830	770 870								
15	1150 1210	1140 1260	1140 1270	1140 1280							
20	1530 1600	1520 1630	1520 1640	1560 1670	1520 1750						
25	1880 2010	1890 2020	1900 2050	1900 2080	1890 2120	1890 2120					
30	2290 2400	2260 2410	2290 2440	2250 2480	2260 2520	2280 2810	2270 2560				
35	2700 2830	2650 2830	2680 2880	2650 2880	2660 2900	2650 2940	2670 2990	2700 3020			
40	3100 3180	3030 3220	3030 3280	3060 3300	3020 3500	3170 3330	3020 3360	3060 3410	3050 3430		
45	3440 3630	3470 3650	3380 3650	3420 3650	3410 3690	3450 3740	3400 3760	3400 3780	3450 3860	3400 3840	
50	3800 4000	3800 4040	3810 4090	3810 4070	3790 4120	3780 4430	3780 4150	3790 4250	3780 4240	3770 4230	3940 4290

Figure: Time in milliseconds for `GenShare`, at the 128-bit security level, for various k and n. Top numbers in each cell are for our scheme; bottom numbers are for [Schoenmakers 99].

Performance: Verification

128 bit	$k = 1$	5	10	15	20	25	30	35	40	45	50
$n = 10$	990 690	1050 780	1280 1120								
15	1510 1050	1550 1150	1770 1510	2170 2130							
20	1980 1390	2040 1490	2310 1880	2700 2510	3280 3510						
25	2470 1760	2530 1860	2740 2230	3190 2930	3770 3900	4590 5230					
30	3020 2090	3020 2230	3240 2620	3640 3340	4250 4410	5040 5680	6060 7430				
35	3520 3020	3560 2600	3780 3030	4200 3750	4760 4830	5570 6220	6560 7940	8380 10060			
40	4030 2770	4070 2910	4340 3410	4670 4210	5280 5350	6140 6740	7030 8550	8290 10800	9640 13550		
45	4480 3150	4520 3300	4720 3860	5160 4600	5790 5800	6870 7300	7550 9210	8730 11350	10210 14000	11700 16990	
50	4960 3480	5140 3670	5410 4200	5610 5200	6220 6270	7020 7930	8030 9810	9210 12260	10580 14930	12200 17960	14240 21640

Figure: Time in milliseconds for `Verify`, at the 128-bit security level, for various k and n. Top numbers in each cell are for our scheme; bottom numbers are for [Schoenmakers 99].

Performance: Reconstruction

128 bit	$k = 1$	5	10	15	20	25	30	35	40	45	50
$n = 10$	20 10	220 90	1020 440								
15	20 10	220 100	980 410	2420 1010							
20	10 10	220 100	1000 420	2370 990	4410 1890						
25	20 0	220 110	1000 420	2390 990	4330 1850	7080 2930					
30	10 10	220 90	990 420	2350 980	4350 1850	6930 2910	10360 4300				
35	10 0	250 100	1020 430	2400 990	4460 1820	6990 2900	10360 4250	14230 5920			
40	20 10	210 90	1000 430	2350 1000	4340 1840	6960 2900	10190 4230	14120 5880	18830 7960		
45	10 10	230 110	980 410	2360 1000	4330 1800	7120 2890	10150 4230	14110 5830	18680 7710	24030 9900	
50	10 0	240 100	990 430	2380 980	4350 1830	6980 2930	10240 4220	14050 5900	18620 7820	23920 9900	30170 12510

Figure: Time in milliseconds for `Reconst`, at the 128-bit security level, for various k and n. Top numbers in each cell are for our scheme; bottom numbers are for [Schoenmakers 99].