Enforcing Integrity of Execution in Distributed Workflow Management Systems [*]

Frederic Montagut
SAP Labs France, Institut Eurecom
805, Avenue du Docteur Maurice Donat
Font de l'Orme, 06250 Mougins, France
frederic.montagut@sap.com

Refik Molva
Institut Eurecom
2229 Route des Cretes
06904 Sophia-Antipolis, France
refik.molva@eurecom.fr

Abstract

As opposed to centralized workflow management systems, the distributed execution of workflows can not rely on a trusted centralized point of coordination. As a result, this flexible decentralized setting raises specific security requirements, such as the compliance of the overall sequence of operations with the pre-defined workflow execution plan, that are not yet met by existing decentralized workflow infrastructures. In this paper, we propose new security mechanisms capitalizing on onion encryption techniques and security policy models in order to assure the integrity of the distributed execution of workflows and to prevent workflow instance forging to name a few features. These mechanisms can easily be integrated into distributed workflow management systems as our design is strongly coupled with the runtime specification of decentralized workflows.

1. Introduction

Distributed workflow management systems [3, 8, 13] eliminate the need for a centralized coordinator that can be a performance bottleneck in some business scenarios. This flexibility introduced by decentralized workflows on the other hand raises new security requirements like integrity of workflow execution in order to assure the compliance of the overall sequence of operations with the pre-defined workflow execution plan. As opposed to usual centralized workflow management systems, the distributed execution of workflows can not indeed rely on a trusted centralized coordination mechanism to manage the most basic execution primitives such as message routing between business partners. Yet, existing decentralized workflow management systems appear to be limited when it comes to integrating security mechanisms that meet these specific requirements in addition to the ones identified in the centralized setting. Even though some recent research efforts in the field of distributed workflow security have indeed been focusing on issues related to the management of rights in business partner assignment or detecting conflicts of interest [1, 7, 10] basic security issues related to the security of the overall workflow execution such as integrity and evidence of execution have not yet been addressed.

In this paper, we propose new mechanisms supporting the secure execution of workflows in the decentralized setting. These mechanisms, capitalizing on onion encryption techniques [15] and security policy models, assure the integrity of the distributed execution of workflows and prevent business partners from being involved in a workflow instance forged by a malicious peer. Our solution can easily be integrated into the runtime specification of decentralized workflow management systems as illustrated in this paper using the pervasive workflow model specified in [13]. The remainder of the paper is organized as follows. Section 2 and 3 outline the pervasive workflow model and the associated security requirements, respectively. In section 4 our solution is specified while in section 5 the runtime specification of the secure distributed workflow execution is presented. In section 6 the security properties of the mechanisms we designed are validated. Finally section 7 discusses related work and section 8 presents the conclusion.

2. Workflow model

The workflow management system used to support our approach was designed in [13]. This model supports the execution of business processes in environments without infrastructure and features a distributed architecture characterized by two objectives:

- fully decentralized: the workflow management task is carried out by a set of devices in order to cope with the lack of dedicated infrastructure
- dynamic assignment of business partners to workflow tasks: the actors can be discovered at runtime

Having designed an abstract representation of the workflow whereby business partners are not yet assigned to tasks, a partner launches the execution and executes a first set of

[*]This work has been partially sponsored by EU IST Directorate General as a part of FP6 IST project R4eGov and by SAP Labs France S.A.S.

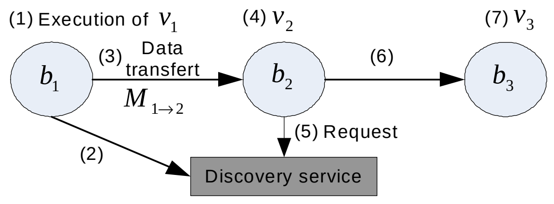


Figure 1. Pervasive workflow runtime

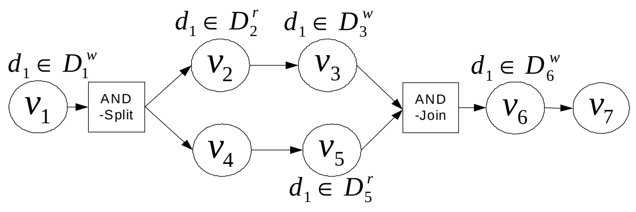


Figure 2. Workflow example

tasks. Then the initiator searches for a partner able to perform the next set of tasks. Once the discovery phase is complete, a workflow message including all data is sent by the workflow initiator to the newly discovered partner and the workflow execution further proceeds with the execution of the next set of tasks and a new discovery procedure. The sequence composed of the discovery procedure, the transfer of data and the execution of a set of tasks is iterated till the final set of tasks. In this decentralized setting, the data transmitted amongst partners include all workflow data. We note W the abstract representation of a distributed workflow defined by $W = \{(v_i)_{i\in[1,n]}, \delta\}$ where v_i denotes a vertex which is a set of workflow tasks that are performed by a business partner from the receipt of workflow data till the transfer of data to the next partner and δ is the set of execution dependencies between those vertices. We note $(M_{i\to j_p})_{p\in[1,z_i]}$ the set of workflow messages issued by b_i to the z_i partners assigned to the vertices $(v_{j_p})_{p\in[1,z_i]}$ executed right after the completion of v_i. The instance of W wherein business partners have been assigned to vertices is denoted $W_b = \{W_{iid}, (b_i)_{i\in[1,n]}\}$ where W_{iid} is a string called workflow instance identifier. This model is depicted in figure 1. In this paper, we only focus on a subset of execution dependencies or workflow patterns namely, SEQUENCE, AND-SPLIT, AND-JOIN, OR-SPLIT and OR-JOIN.

3. Security requirements

As opposed to centralized workflow management systems the distributed execution of workflows raises security constraints due to the lack of a dedicated infrastructure assuring the management and control of the workflow execution. As a result, security features such as compliance of the workflow execution with the pre-defined plan are no longer assured. We group the security requirements we identified for distributed workflow systems into three main categories: authorization, proof of execution and data protection.

3.1. Authorization

The main security requirement for a workflow management system is to ensure that only authorized business partners are assigned to workflow tasks throughout an instance. In the decentralized setting, the assignment of workflow tasks is managed by business partners themselves relying on a service discovery mechanism. In this case, the business partner assignment procedure enforces a matchmaking procedure whereby business partners' security credentials are matched against security requirements for tasks.

3.2. Execution proofs

A decentralized workflow management system does not offer any guarantee regarding the compliance of actual execution of workflow tasks with the pre-defined execution plan. Without any trusted coordinator to refer to, the business partner b_i assigned to the vertex v_i needs to be able to verify that the vertices scheduled to be executed beforehand were actually executed according to the workflow plan. This is a crucial requirement to prevent any malicious peer from forging a workflow instance.

3.3. Workflow data protection

In the case of decentralized workflow execution, the set of workflow data denoted $D = (d_k)_{k\in[1,j]}$ is transferred from one business partner to another. This raises major requirements for workflow data security in terms of integrity, confidentiality and access control as follows:

- data confidentiality: for each vertex v_i, the business partner b_i assigned to v_i should only be authorized to read a subset D_i^r of D
- data integrity: for each vertex v_i, the business partner b_i assigned to v_i should only be authorized to modify a subset D_i^w of D_i^r
- access control: the subsets D_i^r and D_i^w associated with each vertex v_i should be determined based on the security policy of the workflow

4. The solution

4.1. Key management

Two types of key pairs are introduced in our approach. Each vertex v_i is first associated with a policy pol_i defining the set of credentials a candidate partner needs to satisfy in order to be assigned to v_i. The policy pol_i is mapped to a key pair (PK_{pol_i}, SK_{pol_i}) where SK_{pol_i} is the policy private key and PK_{pol_i} the policy public key. Thus satisfying the policy pol_i is equivalent to knowing the private key

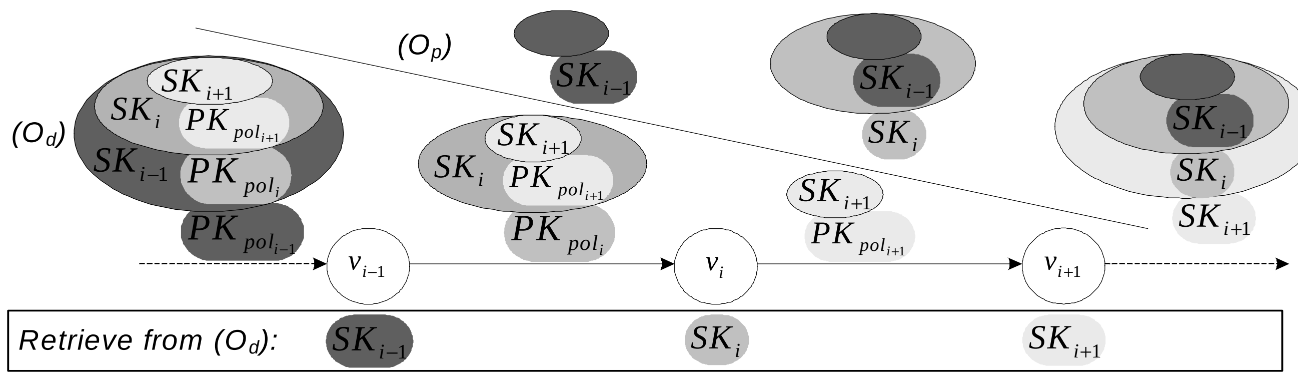


Figure 3. Key management

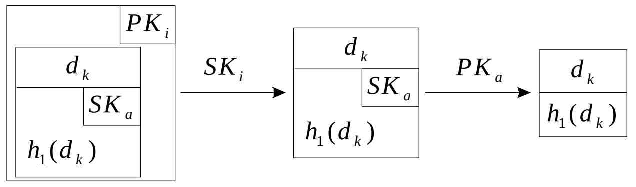


Figure 4. Access to workflow data

SK_{pol_i}. The policy private key SK_{pol_i} can be distributed by a simple key distribution server based on the compliance of business partners with policy pol_i or by means of a more sophisticated cryptographic scheme such as group key distribution [17] or policy-based encryption [2]. Second, we introduce vertex key pairs $(PK_i, SK_i)_{i\in[1,n]}$ to protect the access to workflow data. We suggest a key distribution scheme wherein a business partner b_i whose identity is *a priori* unknown retrieves the vertex private key SK_i upon his assignment to the vertex v_i. Onion encryption techniques with policy public keys PK_{pol_i} are used to distribute vertex private keys. Furthermore, execution proofs have to be issued along with the workflow execution in order to ensure the compliance of the execution with the pre-defined plan. To that effect, we also leverage onion encryption techniques in order to build an onion structure with vertex private keys to assure the integrity of the workflow execution. The suggested key distribution scheme (O_d) and the execution proof mechanism (O_p) are depicted in figure 3 and specified later on in the paper.

In the sequel of the paper, $\mathcal{M}$ denotes the message space, $\mathcal{C}$ the ciphertext space and $\mathcal{K}$ the key space. The encryption of a message $m \in \mathcal{M}$ with a key $K \in \mathcal{K}$ is noted $\{m\}_K$ and h_1, h_2 denote one-way hash functions.

4.2. Data protection

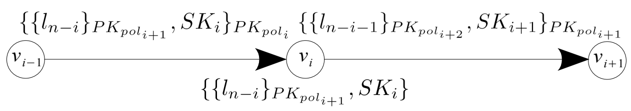
The role of a business partner b_i assigned to a vertex v_i consists in processing the workflow data that are granted read-only and read-write access during the execution of v_i. We define a specific structure depicted in figure 4 called data block to protect workflow data accordingly. Each data block consists of two fields: the actual data d_k and a signature $sign_a(d_k) = \{h_1(d_k)\}_{SK_a}$. We note $B_k^a = (d_k, sign_a(d_k))$ the data block including the data segment d_k that has last been modified during the execution of v_a. The data block B_k^a is also associated with a set of signatures denoted H_k^a that is computed by b_a assigned to v_a. $H_k^a = \big\{\{h_1(\{B_k^a\}_{PK_l})\}_{SK_a} | l \in R_k^a\big\}$ where R_k^a is the set defined as follows. $R_k^a = \{l \in [1,n] | (d_k \in D_l^r)$ and (v_l is executed after v_a) and (v_l is not executed after $v_{p_{(a,l,k)}}$)$\}$ where $v_{p_{(a,l,k)}}$ denotes the first vertex executed after v_a such that $d_k \in D^w_{p_{(a,l,k)}}$ and that is located on the same branch of the workflow as v_a and v_l. For instance, consider the example of figure 2 whereby d_1 is in D_1^w, D_2^r, D_3^w, D_5^r and D_6^w, $v_{(1,2,1)} = v_3$, $R_1^1 = \{2,3,5,6\}$ and $R_1^3 = \{6\}$.

When the business partner b_i receives the data block B_k^a encrypted with PK_i (i.e. he is granted read access on d_k), he decrypts the structure using SK_i in order to get access to d_k and $sign_a(d_k)$. b_i is then able to verify the integrity of d_k using PK_a, i.e. that d_k was last modified after the execution of v_a. Further, if b_i is granted write access on d_k, he can update the value of d_k and compute $sign_i(d_k)$ yielding a new data block B_k^i and a new set H_k^i. If on the contrary b_i receives B_k^a encrypted with PK_m (in this case v_m is executed after v_i), b_i can verify the integrity of $\{B_k^a\}_{PK_m}$ by matching $h_1(\{B_k^a\}_{PK_m})$ against the value contained in H_k^a.

The integrity and confidentiality of data access thus relies on the fact that the private key SK_i is made available to b_i only prior to the execution of v_i. The corresponding distribution mechanism is presented in the next section.

4.3. Vertex private key distribution mechanism

The objective of the vertex private key distribution mechanism is to ensure that only the business partner b_i assigned to v_i at runtime and whose identity is *a priori* unknown can access the vertex private key SK_i. The basic idea behind this mechanism is to map the workflow structure in terms of execution patterns with an onion structure O_d so that at each step of the workflow execution a layer of O_d is peeled off using SK_{pol_i} and SK_i is revealed.



$\{\{l_{n-i}\}_{PK_{pol_{i+1}}}, SK_i\}_{PK_{pol_i}}$
$\{\{l_{n-i-1}\}_{PK_{pol_{i+2}}}, SK_{i+1}\}_{PK_{pol_{i+1}}}$
v_{i-1}
v_i
v_{i+1}
$\{\{l_{n-i}\}_{PK_{pol_{i+1}}}, SK_i\}$

Figure 5. SEQUENCE pattern

Definition 4-1. Let X a set. An onion O is a multi-layered structure composed of a set of n subsets of X $(l_k)_{k\in[1,n]}$, such that $\forall k \in [1,n]\ l_k \subseteq l_{k+1}$. The elements of $(l_k)_{k\in[1,n]}$ are called layers of O, in particular, l_1 and l_n are the lowest and upper layers of O, respectively. We note $l_p(O)$ the layer p of an onion O.

Definition 4-2. Let $A = (a_k)_{k\in[1,j]}$ and $B = (b_k)_{k\in[1,l]}$ two onion structures, A is said to be wrapped by B, when $\exists k \in [1,l]$ such that $a_j \subseteq b_k$.

We first present how vertex private keys are distributed to partners with respect to various workflow patterns including SEQUENCE, AND-SPLIT, AND-JOIN, OR-SPLIT and OR-JOIN before describing how those are combined in the execution of a complete workflow.

4.3.1. SEQUENCE workflow pattern. Vertex private keys are sequentially distributed to business partners. In this case, an onion structure assuring the distribution of vertex private keys is sequentially peeled off by partners. Considering a sequence of n vertices $(v_i)_{i\in[1,n]}$ b_1 assigned to v_1 initiates the workflow execution with the onion structure O defined as follows.

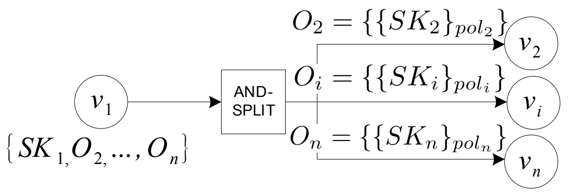
$$O : \begin{cases} l_1 = \{SK_n\} \\ l_i = \left\{\{l_{i-1}\}_{PK_{pol_{n-i+2}}}, SK_{n-i+1}\right\} \text{ for } i \in [2,n] \\ l_{n+1} = \left\{\{l_n\}_{PK_{pol_1}}\right\} \end{cases}$$

The workflow execution further proceeds as depicted in figure 5. For $i \in [2, n-1]$ the business partner b_i assigned to the vertex v_i receives $\{l_{n-i+1}(O)\}_{PK_{pol_i}}$, peels one layer off by decrypting it using SK_{pol_i}, reads $l_{n-i+1}(O)$ to retrieve SK_i and sends $\{l_{n-i}(O)\}_{PK_{pol_{i+1}}}$ to b_{i+1}.

4.3.2. AND-SPLIT workflow pattern. In the case of the AND-SPLIT pattern, the business partners $(b_i)_{i\in[2,n]}$ assigned to the vertices $(v_i)_{i\in[2,n]}$ are contacted concurrently by b_1 assigned to the vertex v_1. In this case, $n-1$ vertex private keys should be delivered to $(b_i)_{i\in[2,n]}$ and the upper layer of the onion O_1 available to b_1 therefore wraps SK_1 and $n-1$ onions $(O_i)_{i\in[2,n]}$ to be sent to $(b_i)_{i\in[2,n]}$ as depicted in figure 6.

$$\begin{array}{l} O_1 = \{SK_1, O_2, O_3, .., O_n\} \\ O_i = \left\{\{SK_i\}_{PK_{pol_i}}\right\} \text{ for } i \in [2,n] \end{array}$$

4.3.3. AND-JOIN workflow pattern. Since there is a single workflow initiator, the AND-JOIN pattern is preceded in the workflow by an AND-SPLIT pattern. In this case, the vertex v_n is executed by the business partner b_n if and only if the latter receives $n-1$ messages as depicted in figure 7.



$O_2 = \{\{SK_2\}_{pol_2}\}$
v_2
v_1
AND-SPLIT
$O_i = \{\{SK_i\}_{pol_i}\}$
v_i
$\{SK_1, O_2, ..., O_n\}$
$O_n = \{\{SK_n\}_{pol_n}\}$
v_n

Figure 6. AND-SPLIT pattern

The vertex private key SK_n is thus divided into $n-1$ parts and defined by $SK_n = SK_{n_1} \oplus SK_{n_2} \oplus ... \oplus SK_{n_{n-1}}$. The onion O_i sent by b_i thus includes SK_{n_i}. Besides, in order to avoid redundancy, the onion structure λ associated with the sequel of the workflow execution right after v_n is only included in one of the onions received by b_n. Each $(b_i)_{i\in[1,n-1]}$ therefore sends O_i to b_n where

$$\begin{array}{l} O_1 = \left\{\{\lambda, SK_{n_1}\}_{PK_{pol_n}}\right\} \\ O_i = \left\{\{SK_{n_i}\}_{PK_{pol_n}}\right\} \text{ for } i \in [2, n-1] \end{array}$$

4.3.4. OR-SPLIT workflow pattern. This is an exclusive choice, v_1 sends one message to the appropriate participant.

$$\begin{array}{l} O_1 = \{SK_1, O_2, O_3, .., O_n\} \\ O_i = \left\{\{SK_i\}_{PK_{pol_i}}\right\} \text{ for } i \in [2,n] \end{array}$$

O_1 is available to the participant assigned to v_1. This is the same structure as the AND-SPLIT pattern, yet the latter only sends the appropriate O_i to v_i depending on the result of the OR-SPLIT condition.

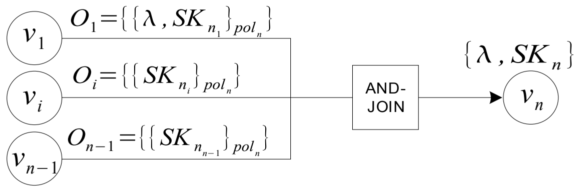
4.3.5. OR-JOIN workflow pattern. Since there is a single workflow initiator, the OR-JOIN is preceded in the workflow by an OR-SPLIT pattern. The partner assigned to v_n receives in any cases a single message thus a single vertex private key is required that is sent by one of the $(b_i)_{[1,n-1]}$ depending on the choice made at the previous OR-SPLIT in the workflow. b_n thus receives in any cases:

$$O = \left\{\{\lambda, SK_n\}_{PK_{pol_n}}\right\}$$

where λ is an onion structure associated with the sequel of the workflow execution right after v_n.

4.3.6. Complete key distribution scheme. The procedure towards building an onion structure corresponding to the workflow structure is rather straightforward and it is only sketched throughout an example. Let's consider the workflow depicted in figure 2. The onion O_d enabling the vertex private keys distribution during the execution of the workflow is defined as follows.

$$\begin{array}{l} O_d = \{\{SK_1, \underbrace{\{SK_2, \{SK_3, \{SK_{6_1}, \overbrace{\{SK_7\}_{PK_{pol_7}}}^{\text{Sequel after} v_6}}}_{\text{First AND-SPLIT branch}} \\ \underbrace{\}_{PK_{pol_6}}\}_{PK_{pol_3}}\}_{PK_{pol_2}}}_{\text{First AND-SPLIT branch}}, \underbrace{\{SK_4, \{SK_5, \{SK_{6_2}}_{\text{Second AND-SPLIT branch}} \\ \underbrace{\}_{PK_{pol_6}}\}_{PK_{pol_5}}\}_{PK_{pol_4}}}_{\text{Second AND-SPLIT branch}}\}_{PK_{pol_1}}\} \end{array}$$



$O_1 = \{\{\lambda, SK_{n_1}\}_{pol_n}\}$
v_1
$O_i = \{\{SK_{n_i}\}_{pol_n}\}$
v_i
$O_{n-1} = \{\{SK_{n_{n-1}}\}_{pol_n}\}$
v_{n-1}
AND-JOIN
$\{\lambda, SK_n\}$
v_n

Figure 7. AND-JOIN pattern

The onions associated with the two branches forming the AND-SPLIT pattern are wrapped by the layer corresponding to v_1. Only the first AND-SPLIT branch includes the sequel of the workflow after v_6.

4.4. Execution proofs

Along with the workflow execution, an onion structure O_{p_i} is built at each execution step i with vertex private keys in order to allow business partners to verify the integrity of the workflow execution. The onion structure is initialized by the business partner b_1 assigned to v_1 who computes $O_{p_1} = \{\{h_1(P_W)\}_{SK_{pol_1}}\}$ where P_W is called workflow policy and is defined as follows.

Definition 4-3. The workflow specification S_W denotes the set $S_W = \{W, (J_i^r, J_i^w, pol_i)_{i\in[1,n]}, h_1\}$ where $J_i^r = \{k \in [1,j] | d_k \in D_i^r\}$ and $J_i^w = \{k \in [1,j] | d_k \in D_i^w\}$ (J_i^r and J_i^w basically specify for each vertex the set of data that are granted read-only and read-write access, respectively). S_W is defined at workflow design phase.

The workflow policy P_W denotes the set $P_W = S_W \cup \{W_{iid}, h_2\} \cup \{PK_i | i \in [1,n]\}$. P_W is a public parameter computed by the workflow initiator b_1 and that is available to the business partners involved in the execution of W.

The onion structure O_p is initialized this way so that it cannot be replayed as it is defined for a specific instance of a workflow specification.

At the step i of the workflow execution, b_i receives $O_{p_{i-1}}$ and encrypts its upper layer with SK_i to build an onion O_{p_i} which he sends to b_{i+1} upon completion of v_i. Considering a set $(v_i)_{[1,n]}$ of vertices executed in sequence we get:

$$\begin{aligned} O_{p_1} &= \{\{h_1(P_W)\}_{SK_{pol_1}}\} \\ O_{p_2} &= \{\{O_{p_1}\}_{SK_2}\} \\ O_{p_i} &= \{\{O_{p_{i-1}}\}_{SK_i}\} \text{ for } i \in [3,n] \end{aligned}$$

The building process of O_{p_i} is based on workflow execution patterns ; yet since it is built at runtime contrary to the onion O_d, this is straightforward. First, there is no specific rule for OR-SPLIT and OR-JOIN patterns. Second, when encountering an AND-SPLIT pattern, the same structure O_{p_i} is concurrently sent while in case of an AND-JOIN, the $n-1$ onions received by a partner b_n are wrapped by a single structure: $O_{p_n} = \{\{O_{p_1}, O_{p_2}, .., O_{p_{n-1}}\}_{SK_n}\}$.

In order to verify that the workflow execution is compliant with the pre-defined plan when he starts the execution

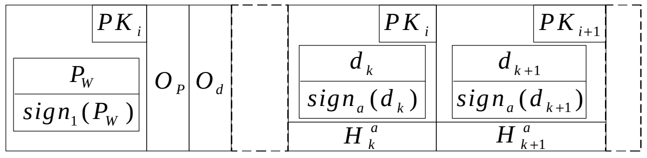


Figure 8. Workflow message structure

of the vertex v_i, the business partner b_i assigned to v_i just peels off the layers of $O_{p_{i-1}}$ using the vertex public keys of the vertices previously executed based on S_W. Doing so he retrieves the value $\{h_1(P_W)\}_{SK_{pol_1}}$ that should be equal to the one he can compute given P_W, if the workflow execution has been so far executed according to the plan.

Considering the example depicted in figure 2, at the end of the workflow execution the onion O_p is defined as follows.

$$O_p = \{\{\underbrace{\{\{\{\{h_1(P_W)\}_{SK_{pol_1}}\}_{SK_2}\}_{SK_3}}_{\text{First AND-SPLIT branch}},\ \underbrace{\{\{\{h_1(P_W)\}_{SK_{pol_1}}\}_{SK_4}\}_{SK_5}}_{\text{Second AND-SPLIT branch}}\}_{SK_6}\}_{SK_7}\}$$

$\{h_1(P_W)\}_{SK_{pol_1}}$ is sent by b_1 assigned to v_1 to both b_2 and b_4 assigned to v_2 and v_4 respectively. The onion structure associated with the two branches forming the AND-SPLIT pattern thus includes $\{h_1(P_W)\}_{SK_{pol_1}}$ twice.

4.5. Vertex key pair generation

Vertex key pairs have to be defined for a single instance of a workflow specification in order to avoid replay attacks. To that effect, we propose to capitalize on ID-based encryption techniques [5] in the specification of the set $(PK_i, SK_i)_{i\in[1,n]}$. For all $i \in [1,n]$ (PK_i, SK_i) is defined by:

$$\begin{cases} PK_i = h_1(W_{iid} \oplus S_W \oplus v_i) \\ SK_i = s \times h_2(PK_i) \end{cases}$$

where $s \in \mathbb{Z}_q^*$ for a prime q. s is called master key and is held by the vertex private key generator [5] who is in our case the workflow initiator.

This vertex key pair specification has a double advantage. First vertex key pairs cannot be reused during any other workflow instance and second vertex public keys can be directly retrieved from W and W_{iid} when verifying the integrity of workflow data or peeling off the onion O_p.

4.6. Communication protocol

In order to support a coherent execution of the mechanisms presented so far, workflow messages exchanged between business partners consist of the set of information that is depicted in figure 8.

Workflow data $(d_k)_{k\in[1,j]}$ are all transported between business partners and satisfy the data block specification. A single message may include several copies of the same data block structure that are encrypted with different vertex public keys based on the execution plan. This can be the case with AND-SPLIT patterns. Besides, workflow data can be stored in two different ways depending on the requirements for the execution. Either we keep the iterations of data resulting from each modification in workflow messages till the end of the execution or we simply replace data content upon completion of a vertex. The bandwidth requirements are higher in the first case since the size of messages increases as the workflow execution proceeds further.

P_W is required to retrieve vertex and policy public keys and specifies the workflow execution plan.

The two onion structures O_d and O_p are also included in the message.

Upon receipt of the message depicted in figure 8 a business partner b_i assigned to v_i retrieves first the vertex private key from O_d. He then checks that P_W is genuine i.e. that it was initialized by the business partner initiator of the workflow assigned to v_1. He is later on able to verify the compliance of the workflow execution with the plan using O_p and finally he can process workflow data.

5. Secure execution of decentralized workflows

In this section we specify how the mechanisms presented so far are combined to support the secure execution of a workflow in the decentralized setting. After an overview of the execution steps, the secure workflow execution is described in terms of the workflow initiation and runtime specifications.

5.1. Execution process overview

Integrating security mechanisms to enforce the security requirements of the decentralized workflow execution requires a process strongly coupled with both workflow design and runtime specifications. At the workflow design phase, the workflow specification S_W is defined in order to specify for each vertex the sets of data that are accessible in read and write access and the credentials required by potential business partners to be assigned to workflow vertices. At workflow initiation phase, the workflow policy P_W is specified and the onion O_d is built. The workflow initiator builds then the first set of workflow messages to be sent to the next partners involved. This message generation process consists of the initialization of the data blocks and that of the onion O_p.

At runtime, a business partner b_i chosen to execute a vertex v_i receives a set of workflow messages. Those messages are processed to retrieve SK_i from the onion O_d and to access workflow data. Once the vertex execution is complete b_i builds a set of workflow messages to be dispatched to the next partners involved in the execution. In this message building process, the data and the onion O_p are updated.

The set of functional operations composing the workflow initiation and runtime specifications is precisely specified later on in this section. In the following N_k^i denotes the set defined by $N_k^i = \{l \in [1,n] | d_k \in D_l^r$ and v_l is executed right after $v_i\}$. Consider the example of figure 2: d_1 is accessed during the execution of the vertices v_1, v_2 and v_5 thus $N_1^1 = \{2, 5\}$.

5.2. Workflow initiation

The workflow is initiated by the business partner b_1 assigned to the vertex v_1 who issues the first set of workflow messages $(M_{1\to j_p})_{p\in[1,z_1]}$. The workflow initiation consists of the following steps.

1. Workflow policy specification: generate $(PK_i, SK_i)_{i\in[1,n]}$
2. Initialization of the onion O_d
3. Data block initialization: compute $\forall k \in [1,j]$ $sign_1(d_k)$
4. Data block encryption: $\forall k \in [1,j]$ determine N_k^1 and compute $\forall k \in [1,j], \forall l \in N_k^1$ $\{B_k^1\}_{PK_l}$
5. Data block hash sets: $\forall k \in [1,j]$ determine R_k^1 and compute $\forall k \in [1,j], \forall l \in R_k^1$ $\{h_1(\{B_k^1\}_{PK_l})\}_{SK_1}$
6. Initialization of the onion O_p: compute O_{p_1}
7. Message generation based on W and $(N_k^1)_{k\in[1,j]}$

The steps one and two are presented in sections 4.5 and 4.3, respectively. The workflow messages are generated with respect to the specification defined in figure 8 and sent to the next business partners involved. This includes the initialization of the onion O_p and that of data blocks which are encrypted with appropriate vertex public keys.

5.3. Workflow message processing

A business partner b_i being assigned to a vertex v_i proceeds as follows upon receipt of the set of workflow messages $(M_{j_p\to i})_{p\in[1,k_i]}$ sent by the k_i business partners assigned to the vertices $(v_{j_p})_{p\in[1,k_i]}$ executed right before v_i.

1. Retrieve SK_i from O_d
2. Data block decryption with SK_i based on J_i^r
3. Execution proof verification: peel off the onion O_p
4. Data integrity check based on W and P_W
5. Vertex execution
6. Data block update: compute $\forall k \in J_i^w$ $sign_i(d_k)$ and update d_k content
7. Data block encryption: $\forall k \in J_i^r$ determine N_k^i and compute $\forall k \in J_i^r, \forall l \in N_k^i$ $\{B_k^i\}_{PK_l}$
8. Data block hash sets: $\forall k \in J_i^w$ determine R_k^i and compute $\forall k \in J_i^w, \forall l \in R_k^i$ $\{h_1(\{B_k^i\}_{PK_l})\}_{SK_i}$
9. Onion O_p update: compute O_{p_i}
10. Message generation based on W and $(N_k^i)_{k\in[1,j]}$

After having retrieved SK_1 from O_d, b_i verifies the integrity of workflow data and that the execution of the workflow up to his vertex is consistent with the onion O_p. Workflow data are then processed during the execution of v_i and data blocks are updated and encrypted upon completion. Finally b_i computes O_{p_i} and issues the set of workflow messages $(M_{i \to j_p})_{p \in [1, z_i]}$ to the intended business partners.

6. Security

There are several alternatives with respect to the management of the key pair (PK_{pol_i}, SK_{pol_i}), including simple key distribution based on the policy compliance, group key management or policy-based cryptography. Amongst those alternatives, we only discuss the policy based cryptography scenario as part of the security evaluation of our solution. In the following, we make two assumptions:

- IND-PB-CCA: the policy-based encryption scheme used in the specification of $(PK_{pol_i}, SK_{pol_i})_{[1,n]}$ is semantically secure against a chosen ciphertext attack for policy-based encryption and the associated policy-based signature scheme achieves signature unforgeability [2]
- IND-CCA: the public key encryption scheme used in the specification of $(PK_i, SK_i)_{[1,n]}$ is semantically secure against a chosen ciphertext attack the associated signature scheme achieves signature unforgeability

Claim 6-1. *The integrity of the distributed workflow execution is ensured. This basically means that workflow data are accessed and modified by authorized business partners based on the pre-defined plan specified by means of the sets J_i^r and J_i^w.*

Proof: This property is ensured by the onion O_d which assures the vertex key distribution used in the access to workflow data based on the workflow execution plan.

Assuming that a workflow initiator builds O_d based on the methodology specified in 4.3 and under IND-PB-CCA, we claim that it is not feasible for an adversary $\mathcal{A}$ to extract the vertex private key SK_i from O_d if $\mathcal{A}$ does not satisfy the set of policies $(pol_{i_k})_{k \in [1,l]}$ associated with the set of vertices $(v_{i_k})_{k \in [1,l]}$ executed prior to v_i in W. This is true as the structure of O_d is mapped to W.

Claim 6-2. *Upon receipt of a workflow message, a business partner is sure that the workflow has been properly executed so far provided that he trusts the business partners satisfying the policy pol_1.*

Proof: This means that an adversary that does not verify a policy that is trusted by some business partners can not forge a workflow instance, i.e. that he can not produce a workflow message faking a valid workflow instance. This property is enforced by the onion O_p.

Assuming that a workflow initiator builds O_p based on the methodology specified in 4.4 and under IND-PB-CCA, we claim that the onion structure O_p is unforgeable. To assure the unforgeability property, we need to verify that:

1. a genuine onion structure O_p built during a previous instance of a workflow can not be replayed
2. an onion structure O_p can not be built by an adversary that is not trusted by business partners

The first property is enforced by the fact that an onion structure O_p properly built by trustworthy peers is bound to a specific workflow policy P_W and thus can not be reused during an attempt to execute a malicious workflow instance. The second property is straightforward under IND-PB-CCA as the policy-based signature scheme achieves signature unforgeability. Thus an adversary can not produce a valid onion $O_{p_1} = \left\{ \{h_1(P_W)\}_{SK_{pol_1}} \right\}$.

Claim 6-3. *Assuming that business partners involved in a workflow instance do not share vertex private keys they retrieve from the onion O_d, our solution achieves the following data integrity properties:*

- *Data truncation and insertion resilience: any business partner can detect the deletion or the insertion of a piece of data in a workflow message*
- *Data content integrity: any business partner can detect the integrity violation of a data block content in a workflow message*

Proof: The first property is ensured as the set of workflow data blocks that should be present in a workflow message is specified in P_W, the workflow message formatting has thus to be compliant with the workflow specification. The second property is assured by the fact that an adversary can not modify a given data block without providing a valid signature on this data block. This property relies on the unforgeability of the signature scheme used in the data block and hash set specifications.

These three security properties enable a coherent and secure execution of distributed workflows, yet our solution can still be optimized to avoid the replication of workflow messages. A business partner may indeed send the same workflow message several times to different partners satisfying the same security policy resulting in concurrent executions of a given workflow instance. A solution based on a stateful service discovery mechanism can be envisioned to cope with this problem.

7. Related work

Security of cross-organizational workflows in both centralized and decentralized settings has been an active research field over the past years mainly focusing on access control, separation of duty and conflict of interests [4, 9, 10] issues. However, in the decentralized setting issues related to the integrity of workflow execution and workflow instance forging, which are tackled in our paper have been left aside. In [7, 1] mechanisms are proposed for the management of conflicts of interest [6] during the distributed execution of workflows. These pieces of work specify solutions in the design of access control policies to prevent business partners from accessing data that are not part of their classes of interest. These approaches do not address the issue of policy enforcement with respect to integrity of execution in fully decentralized workflow management systems. Nonetheless, the access control policy models suggested in [7, 1] can be used to augment our work especially in the specification of the sets J_i^r and J_i^w at workflow design time.

Onion encryption techniques have been introduced in [15] and are widely used to enforce anonymity in network routing protocols [11] or mobile agents [12]. In our approach, we map onion structures with workflow execution patterns in order to build proofs of execution and enforce access control on workflow data. As a result, more complex business scenarios are supported by our solution than usual onion routing solutions. Furthermore, combined with policy encryption techniques, our solution provides a secure runtime environment for the execution of fully decentralized workflows supporting runtime assignment of business partners, a feature which had not been tackled so far.

Finally, our approach is suitable for any business scenarios in which business roles can be mapped to security policies that can be associated with key pairs. It can thus be easily integrated into existing security policy models such as chinese wall [6] security model.

8. Conclusion

We presented mechanisms towards meeting the security requirements raised by the execution of workflows in the decentralized setting. Our solution, capitalizing on onion encryption techniques and security policy models, protects the access to workflow data with respect to the pre-defined workflow execution plan and provides proofs of execution to business partners. Those mechanisms can easily be integrated into the runtime specification of decentralized workflow management systems and are further suitable for fully decentralized workflow supporting the runtime assignment of business partners to workflow tasks. We believe that the mechanisms underpinning our approach will foster the development of dynamic business applications whereby workflow actors do not need to rely on a dedicated infrastructure to provide their resources as one of the major flaws slowing down this trend was the lack of security.

References

[1] V. Atluri, S. A. Chun, and P. Mazzoleni. A chinese wall security model for decentralized workflow systems. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 48–57, 2001.

[2] W. Bagga and R. Molva. Policy-based cryptography and applications. In *FC' 2005, 9th International Conference on Financial Cryptography and Data Security, Roseau, The Commonwealth of Dominica*, Mar 2005.

[3] D. Barbara, S. Mehrotra, and M. Rusinkiewicz. Incas: Managing dynamic workflows in distributed environments. *Journal of Database Management*, 7(1), 1996.

[4] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.*, 2(1):65–104, 1999.

[5] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, CA, USA*, pages 213–229, 2001.

[6] D. F. C. Brewer and M. J. Nash. The chinese wall security policy. In *IEEE Symposium on Security and Privacy*, pages 206 –214, 1989.

[7] S.-C. Chou, A.-F. Liu, and C.-J. Wu. Preventing information leakage within workflows that execute among competing organizations. *J. Syst. Softw.*, 75(1-2):109–123, 2005.

[8] A. Cichocki and M. Rusinkiewicz. Providing transactional properties for migrating workflows. *Mob. Netw. Appl.*, 9(5):473–480, 2004.

[9] P. C. K. Hung and K. Karlapalem. A secure workflow model. In *ACSW Frontiers '03: Proceedings of the Australasian information security workshop conference on ACSW frontiers*, pages 33–41, 2003.

[10] M. H. Kang, J. S. Park, and J. N. Froscher. Access control mechanisms for inter-organizational workflow. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 66–74, 2001.

[11] J. Kong and X. Hong. Anodr: anonymous on demand routing with untraceable routes for mobile ad-hoc networks. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 291–302, 2003.

[12] L. Korba, R. Song, and G. Yee. Anonymous communications for mobile agents. In *MATA '02: Proceedings of the 4th International Workshop on Mobile Agents for Telecommunication Applications*, pages 171–181, London, UK, 2002. Springer-Verlag.

[13] F. Montagut and R. Molva. Enabling pervasive execution of workflows. In *Proceedings of the 1st IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom*, 2005.

[14] M. G. Nanda and N. Karnik. Synchronization analysis for decentralizing composite web services. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 407–414, 2003.

[15] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy*, pages 44–54, USA, 1997.

[16] A. R. Tripathi, T. Ahmed, and R. Kumar. Specification of secure distributed collaboration systems. In *ISADS '03: Proceedings of the The Sixth International Symposium on Autonomous Decentralized Systems (ISADS'03)*, page 149, 2003.

[17] C. K. Wong, M. G. Gouda, and S. S. Lam. Secure group communications using key graphs. In *Proceedings of the ACM SIGCOMM '98 conference*, pages 68–79, 1998.