W/CMS: Open-ended Web-based Contract Management System

SHARIL TUMIN
University of Bergen
IT Dept
P.O. Box 7800, 5020 Bergen
NORWAY
edpst@it.uib.no

SYLVIA ENCHEVA
Stord/Haugesund University College
Faculty of Technology, Business and Maritime Sciences
Bjørnsonsg. 45, 5528 Haugesund
NORWAY
sbe@hsh.no

Abstract: On-line contract management framework provides support to contract process flow without the need for all actors to be synchronously present in respect to both time and space. Different contractual processes are managed by the application securely by analyzing data-flow between actors, ushering actors to perform their duties at a timely manner and employing appropriate cryptographic techniques on every step of the way. The implementation must deliver a management system that provides operational properties of authenticity, privacy, trustworthy, reliability, verifiability, and linkability.

Key-words: Contract, Contract Management, Applications security, Secure Web-based Application, Secure Work flow Modelling, Applied Cryptography

1 Introduction

Generally speaking, a contract is a legally binding agreement between two or more parties by which rights are acquired by one or more to act or forbearance on the part of the other or others. A person can act on the behalves of a group of people, organization, cooperation or enterprise. A contract is to be agreed upon. In case of disputes at a later time, a record of the agreement is to be kept and can be used as evident for resolving conflicts. So, a contract is finalized when the content of the contract is agreed and the record of the agreement is accepted trustfully by all concerning parties. Any framework for contract management, digital or otherwise is to promote this trust.

The goal of this paper is to propose an open-ended Web-based Contract Management System (CMS) for deployment where contractual processes can be done between actors using Web-based applications as assertive actions and emails as event triggers. Any registered user can initiate a contract procedure. Anyone with an email account anywhere can register to the system. Any of the registered users can be used as a witness. This proposed framework for on-line contract management will provide the support of contract process flow without the need for the different related actors within the process to be synchronously present with respect to both time and space. Virtual meetings are done by message passing via email or Web-based drop-box areas and actions are executed or triggered for assertion by applications in a timely fashion dictated by rule based process flow directives.

A contract can be: unilateral - f.ex. a will, a testament, bilateral - f.ex. a buying and selling agreement, or multilateral - f.ex. a project team distribution agreement of roles, responsibilities, and resource divisions.

Different types of contracts define different types objects and attributes, both contract specific and domain specific, and these are contract-core objects and contract-core attributes, and related to these contract-core are domain-specific objects and domain-specific attributes. For a particular contract, these objects and attributes together with contract meta data of hashes and digital signatures will be used to produce a finalized contract document in what we call a contract synopsis. The contract synopsis is represented in XML (Extensible Markup Language).

Different types of contract will have different discreet stages. Stages are arranged in an ordered sequence. The system will track these sequences and inform all concerned parties about the status of a current stage. The currently active actors at a particular stage will be ushered to perform certain imperative actions using Web-based applications.

Most contracts will pass through the stages in the following sequence: S1 - Initialization, S2 - Negotiation, S3 - Agreement, S4 - Signing, S5 - Archive (after sealing).

The contract management system needs to facilitate users in 1) initialization phase (Initialization); 2) intermediate phase (Negotiation and Agreement); and 3) finalize phase (Signing and Archive).

In the case of unilateral contract, the system will
provide similar services as given by a notary public officer. The Negotiation and Agreement are not needed for unilateral contract. One service that is not connected directly to wills or testaments but is similar in its processes is related to notarial documents (signed and witnessed notices) of any kind. This type of contracts are useful for attaching timestamps and witnesses on any type of documents, e.g., a scientific paper. Both bilateral and multilateral contracts will have to go through Negotiation and Agreement.

Essentially, actors in the contractual procedure are having different roles with different responsibilities and duties. These roles are: 1) initiator, 2) provider, 3) consumer, 4) witness and 5) archiver. These actors are actively involved in the different stages mentioned above. The implementation must deliver a management system that provides operational properties of 1) authentication; 2) privacy; 3) trustworthy; 4) reliable; and 5) verifiable.

2 Background

2.1 Actors’ Roles and Responsibilities

There are five types of human actor identified in the W/CMS. They are labeled as follows: \( I \) - Initiator, which also will act as intermediary, coordinator, and overseer; \( P \) - Provider, for example seller and benefactor; \( C \) - Consumer, for example client, buyer and beneficiary; \( W \) - Witness; and \( A \) - Archiver.

Figure 1 shows relationships of actors, objects, actions and states. Actors are collected into three types of group, 1) system operators group \( U^M \); 2) active users group \( U^A \); and 3) passive users group \( U^P \). An actor belongs to a set of users of the system \( U \). In order to be a user, a person needs to be registered to the W/CMS. Each user is given a unique username and a password during the registration process.

The username and password will be used for authentication during the subsequent interactions with the system.

Both initiators \( I \) and archivers \( A \) belong to the W/CMS managers groups, \( I \cup A = U^M \). They manage the human side of the W/CMS and do all manual processes concurrent to the automatic processes defined in the system.

An initiator \( I, i \in [1, n] \) is naturally a trusted operator in the W/CMS. Her work is to initialize all pertinence contract-core objects, domain-specific objects and their related attributes. The initiator is prompted into action by a request done by an active user of the system.

Within an extend of a contract forming process, a user is either active or passive, and can never be both at the same time, \( U^A \cap U^P = \emptyset \). Both providers \( P \) and consumers \( C \) belong to the active user group. While witnesses \( W \) belong to the passive user group, \( P \cup C \cup W \subseteq U \). Within a contract forming instance, a provider or a consumer can not also be a witness. All active users \( P_i, i \in [1, m] \) need to provide the W/CMS with supporting documents, conduct negotiations and actively agree to the final agreement of the contract. Potential witnesses, \( W_k, k \in [1, l], W_k \neq P_j \) will be invited to be witnesses and sign a finalized agreement. A witness will sign a particular contract blindly or otherwise, depending on the privacy constraint of the contract. We propose a witness policy plan as shown in Table 1.

For multilateral contract, there is no need of an external witnesses, where the active members of the contract themselves serve as witnesses. In this proposal, we make the assumption that the witnessing process incur significant cost to the contract formation processes, in terms of risks, economy and efforts.

2.2 Operational Properties

As mentioned in passing in Section 1, in order to achieve security and quality criteria, thus promote users’ trust toward the system, the proposed W/CMS must perform with the following operational proper-

![Figure 1: Conceptual CMS use-case](image)

<table>
<thead>
<tr>
<th>Contract type</th>
<th>( P ) or ( C )</th>
<th>( W(\text{min}) )</th>
<th>( W(\text{max}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>unilateral</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>bilateral</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>multilateral</td>
<td>( \geq 3 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Contract type vs Number of Witness
2.3 Design Principles

Effective design connects acting to thinking which in turn connects implementation to formulation. To be effective a design need to be guided by some design principles. These principles provide the designers and the implementers with a base line on which the product as a whole is aiming to achieve when it is put into production. Thus the system that facilitate the contract management, must above all, be usable, adaptable, and manageable. The system users interfaces and work flows must conform to users needs and expectations, logically laid out and simple to understand. There should not be any ambiguities. Regulations regarding different aspects of contract making will change with time, so the system needs to adapt to the current regulatory demands. To be agile and responding to changes, the system and its subsystems need to be managed with manageable interfaces. Therefore, in this paper we propose an open-ended Web-based contract management system guided by these five design principles: Q1 - Usability, Q2 - Adaptability, Q3 - Scalability, Q4 - Interoperability, Q5 - Manageability.

It goes without saying that these five design principles must include and implement the five operational properties mentioned earlier in Section 2.2.

![Figure 2: W/CMS Subsystems use-case](image)

From the conceptual use-case diagram of Figure 1 and the three phases mentioned in Section 1, the working model of the W/CMS can be decomposed into three separate subsystems as shown in Figure 2, which represents a simplified version of process flow schematic of the whole system.

### 2.4 Contract Making Process Flow

Different human actors and their interactions with the system are clearly labelled in Figure 2.

The three main subsystems are 1) users management; 2) contracts formalization; and 3) contracts finalization. They are modeled to be distributed, cooperative and loosely coupled independent subsystems. To promote higher security level, these subsystems are
designed to be operational and administrative independent of each other. System management responsibilities will be given to three non-collaborative teams. These managers will not share administrative secrets and systems’ data.

The users management subsystem manages user registration. A new user will be given a unique username and a self-chosen password, and a pair of RSA [8] public and private keys. The private key is protected by the user password using a symmetric-key encryption. The meta data and private key are also symmetric-key encrypted by the subsystem and send over to the user for safe keeping. The user needs to provide this piece of encrypted text block whenever she changes her password later on.

The contracts formalization subsystem is the bulk of W/CMS. Each contract is initialized when the request is approved by system managers. Each contract will be given a standardized individual database. All contract data and contract documents will be stored in the database. The negotiation will be conducted among persons related to the contract in an environment similar to a blog. All events are logged to this blog by the system software, system managers, coordinator and users. The production of the contract synopsis is done when agreement is reached by all parties. All parties will be asked to sign the contract synopsis.

The contracts finalization subsystem constitutes the finalizing processes for the closing phase of a contract. Witnesses will be called by the archiver to verify the contract synopsis and all participating parties’ signatures. The witnesses will then put their signatures on to the contract synopsis. The archiver will then verify the contract synopsis, all participating parties’ signatures and all witnesses’ signatures, before closing the blog and sealing the contract by putting her signature to the whole contract synopsis and signatures. The signing hierarchy is shown in Figure 3. The sealed contract package and the blog are encrypted and then securely stored.

![Figure 3: Contract Signatures Hierarchy](image)

### 3 Supporting Tools

#### 3.1 Cryptographic Tools

The proposed system employs three simple, well known and widely used cryptographic tools. They are 1) symmetric-key encryption implemented in Blowfish [9]; 2) public-key encryption implemented in RSA [3] public-key cryptography; and 3) cryptographic hash functions implemented by SHA [2] hash algorithms.

Blowfish symmetric-key encryption uses a single secret key. This key is used for both encryption and decryption.

**Blowfish Encryption scheme:**

- **Encryption** $\beta_{enc}$:
  $$c = \beta_{enc}(K_{sec}, m)$$

- **Decryption** $\beta_{dec}$:
  $$m = \beta_{dec}(K_{sec}, c)$$

- **Inverse transformation**:
  $$m = \beta_{dec}(K_{sec}, (\beta_{enc}(K_{sec}, m)))$$

Symmetric-key cryptography is not practical for secure communication between many persons due to key distribution and key management problems. For private communication between two persons, one secret key is enough, but for more than two, say four persons, then six keys are needed and everyone must have three keys each. In fact for $n$ persons, the total number of keys needed are $n!/2 \times (n - 2)!$ and each need to keep $n - 1$ keys.

The utilization of public-key cryptography will solve these problems since each person needs only to have a pair of keys, namely the public and private keys. The public key $K_{pub}$ is readable by all interested parties while the private key $K_{prv}$ is kept secret and only known to the owner.

A public-key cryptographic system depends heavily on computational complexity theory and number theory. RSA [8] is the most well known and widely used cryptographic system in today's digital world. RSA supports non-symmetric-key cryptographic schemes for 1) encryption/decryption; 2) sign/verify; and 3) blind/unblind.

**RSA Encryption scheme:**

- **Encryption** $\xi_{enc}$
  $$c = \xi_{enc}(K_{pub}, m)$$

- **Decryption** $\xi_{dec}$
  $$m = \xi_{dec}(K_{prv}, c)$$

- **Inverse transformation**:
  $$m = \xi_{dec}(K_{prv}, (\xi_{enc}(K_{pub}, m)))$$

**RSA Signature scheme:**
Signing - $\zeta_{\text{sig}}$  
signature, $s = \zeta_{\text{sig}}(K_{\text{prv}}, m)$

Verification - $\zeta_{\text{ver}}$  
verify, $v = \zeta_{\text{ver}}(K_{\text{pub}}, s)$

Inverse transformation:  
$m = \zeta_{\text{ver}}(K_{\text{pub}}, \zeta_{\text{sig}}(K_{\text{prv}}, m))$

The blinding factor $\psi$ is only known to the message owner. Blind signature is useful when anonymity is important [4]. Using blind/unblind, a signee can sign the message without knowing what it contains.

RSA Blind Signature scheme:-

$\text{Blind} - \lambda_{\text{blind}}$  
blind, $b = \lambda_{\text{blind}}(K_{\text{pub}}, \psi, m)$

$\text{Signing} - \zeta_{\text{sign}}$  
signature, $bs = \zeta_{\text{sign}}(K_{\text{prv}}, b)$

$\text{Unblind} - \lambda_{\text{unblind}}$  
unblind, $s = \lambda_{\text{unblind}}(K_{\text{pub}}, \psi, bs)$

$\text{Verification} - \zeta_{\text{ver}}$  
verify, $v = \zeta_{\text{ver}}(K_{\text{pub}}, s)$

Please refer to RFC3447 [3] for a more detailed discussion on Public-Key Cryptography Standards (PKCS) #1 v2.1 by RSA Laboratories.

### 3.2 Software Tools

The system is implemented as a multi-tiers Web-based application, built using free and open source software. The users using their Web browsers will interact with an Apache [1] Web server using HTTPS (Hypertext Transfer Protocol Secure) communication protocol. The programmable environment and the middlewares are written in Python [6]. The back-end database is implemented using SQLite [5].

The Python programming environment is used as integrating middleware between the Web server frontend and the database backend. Some of the important Python packages used to implement the system were 1) mod_python - live-programmable module to Apache; 2) Crypto - cryptographic libraries; 3) xmlrpcpplib - XML-based RPC (Remote Procedure Call); 4) tsklite - SSL v3 (Secure Sockets Layer) and TLS v1 (Transport Layer Security) libraries; and 5) standard Python libraries for examples - SocketServer, BaseHTTPServer, sqlite3, base64 and binascii.

### 4 System

As mentioned earlier in Section 2, the W/CMS can be broken down into three logically separate subsystems as shown in Figure 2. Building on top of these decompositions, the proposed system will support these main operational functionalities as shown in Table 2.

<table>
<thead>
<tr>
<th>Subsystems</th>
<th>Web user interfaces</th>
<th>Utility functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users Management</td>
<td>W1 registration</td>
<td>U1 userSession</td>
</tr>
<tr>
<td></td>
<td>W2 login</td>
<td>U2 keyGen</td>
</tr>
<tr>
<td></td>
<td>W3 changePwd</td>
<td>U3 keyReset</td>
</tr>
<tr>
<td>Contracts Formalization</td>
<td>W2 login</td>
<td>S1 userSession</td>
</tr>
<tr>
<td></td>
<td>W4 contractReq</td>
<td>S2 authorization</td>
</tr>
<tr>
<td></td>
<td>W5 contractNego</td>
<td>C1 contractInit</td>
</tr>
<tr>
<td></td>
<td>W6 contractAgre</td>
<td>C2 blogUtil</td>
</tr>
<tr>
<td></td>
<td>W7 contractWitm</td>
<td>C3 signingUtil</td>
</tr>
<tr>
<td></td>
<td>W8 contractClose</td>
<td>C4 finalize</td>
</tr>
<tr>
<td></td>
<td>W9 reportReq</td>
<td>(*)</td>
</tr>
</tbody>
</table>

These functions are of five types 1) Web-based interfaces - W⋆; 2) user management - U⋆; 3) session and access controller - C⋆; 4) contract management utilities - R⋆; and 5) report, archive and cleanup - R⋆.

Under the registration interface W1 a person will be asked to provide email address. The system will initialize a user in the database and send an activation code to the provided email address. Using this activation code the person will be able to provide the system with her choice of username and password. This will then trigger U1 which will register user data into the database. The U1 will in turn trigger U2 which will produce a pair of public and private keys for the newly registered user. The private key is encrypted using Blowfish encryption with the user’s password as the secret key. The public and encrypted private keys are then securely stored in the system database. The system will then create a user package, which is an encrypted message package containing user data and the keys, and send this message via email to the user for safe keeping.

A registered user can change her password at W3 by providing the system with her email address and activation code. If the system recognizes the given data the system will then request the user to submit a new password together with the user’s user package. The system will then execute U3 which will decrypt the user package and extract user data and private key. If the user data is valid then the system will re-encrypt the private key using the new password and store them.
into the database.

To make use of the facilities provided by the
W/CMS, users must first authenticate themselves at
W2. The two utility functions associated with W2
are S1 and S2. The S1 manages users sessions of all
currently authenticated users. Each user-session con-
contains all necessary information about a particular cur-
rent session, in particular the user current authoriza-
tion and permissions. User-session is directly related
to session Web-cookie.

The W4 provides Web interface to a contract re-
quester and initiator I for submitting request and ap-
proving for a new contract, respectively. Whenever
the request is approved, the initiator I must check that
the provided initial contract data is complete and cor-
rect. Related to W4 is C1. The purpose of C1 is 1) to
initialize a new database with all the contract objects
and attributes, all participating actors U1, and U2,
and 2) to create all the necessary scaffolding for the
new contract management work.

The contract negotiation will be done in an Web
environment similar to a blog at W5. The initiator I
will function as coordinator and a mediator. All U1
will be given access to contact database and environ-
ment. When agreement is reached by all U1, the I will
coordinate the signing and the finalization of the con-
tract using the W6. Related to W6 is C3, the signing
utility function. When the contract synopsis is pro-
duced, an event is triggered on C4.

The C4 will close the blog Web environment and
alert the W and if necessary the witnesses W. By us-
ing the interface W6 the W will function as witness
to the contract synopsis by first verifying the signa-
tures using C5 and then sign using C3. Likewise the
archive A will put her seal by signing using C3 to
the already witnessed contract synopsis. All impor-
tant data particular to the contract will be archived by
A by invoking R2 and their traces are deleted using
R3. A standard report is produced by R1 and send via
email to all U1 as a receipt.

5 Conclusion

A significant portion of this paper is devoted to con-
ceptual discussion on Web-based Contract Man-
agement system. We identified three possible types of
contracts. With a two step user-case diagrams we pro-
pose a system model on loosely coupled system of
three main subsystems. We provide a table showing
functional components of these subsystems. These
functions are further classified into a logical purposes.
All these functions are implemented as modules in our
prototype implementation of the system.

A combination of cryptographic hash functions
(SHA), symmetric-key (Blowfish) and and public-key
(RSA) cryptographic tools provided by hashlib and
Crypto modules in Python made it possible to easily
implement hash calculation, encryption/decryption,
sign/verify and even blind/unblind. There are other
and different cryptographic tools provided by Python
cryptographic libraries.

For the purpose of providing one database per
contract, the SQLite database is more than adequate.
Connectivity between Web front-end to SQLite back-
end database using sqlite3 is robust and easy to utilize.
Blob data type is very useful for storing binary docu-
ments. Each SQLite database is implemented as regu-
lar file in the operating system, so the whole database
can be encrypted as any regular file. Other database
systems may be used instead.

All the software tools used to implement are
open-source and free. The model provides an imple-
mentation structure which is open-ended in the sense
that different tools can be freely mix-and-match.

References:

[1] Apache, The Apache Software Foundation,
HTTP server, http://www.apache.org/,
2009 (last accessed).

(SHS), Federal Information Processing Stan-

Standards (PKCS) #1, RSA Cryptography Spec-

payments, Advances in Cryptology - Crypto ’82,

[5] SQLite, A self-contained, serverless, zero-
configuration, transactional SQL database en-
gine, http://www.sqlite.org/, 2009 (last
accessed).

[6] Python Programming Language,
http://www.python.org/, 2009 (last ac-
cessed).

[7] R. Rivest, The MD5 Message-Digest Algorithm,
MIT Laboratory for Computer Science and RSA

for Obtaining Digital Signatures and Public-Key
Cryptosystems, Communications of the ACM,

[9] B. Schneier, The Blowfish Encryption Algorithm,
http://www.schneier.com/blowfish.html,
2009 (last accessed).