Jikzi: A New Framework for Secure Publishing

Ross J Anderson and Jong-Hyeon Lee

University of Cambridge Computer Laboratory
Pembroke Street, Cambridge CB2 3QG
(rja14,jhl21)@cl.cam.ac.uk

Abstract. We present a new way to deal with security policy on the web. Our aim is to let people build integrated publishing and e-commerce services using appropriate, simple and uniform mechanisms. Our goal is a single framework that applies equally to the publication of catalogues, music, software, public key certificates and even old fashioned books. Our prototype framework, Jikzi, supports multiple security policies – even in the same document – with a single transparent markup language.

Historically, most computer security research was motivated by military concerns; most papers deal with confidentiality, some with authentication. But the emphasis in commerce is on integrity, availability and accountability; confidentiality is often not a major concern at all. This motivates us to take a fresh look at the foundations of our discipline, and revisit not just security policy models but also the authentication and integrity mechanisms we use.

The growing importance of XML may simplify much; if it is adopted as widely as some expect, then everything from digital certificate formats through legacy multilevel secure systems will also be up for review. We believe our prototype system indicates a fruitful alternative approach.

1 Introduction

The security of electronic commerce can be thought of as three separate but loosely related problems – publishing, payment and copy control.

Firstly, most of the activity which people describe as electronic commerce consists of some kind of publishing. Publishing used to be understood as a business sector – the business of making and selling books, CDs, videos and software – but it increasingly makes up the very stuff of online business. Regardless of the product or service sold, electronic commerce consists largely of publishing web pages, catalogues and price lists, as well as more specialist items such as public key certificates.

Secondly, once an online customer has decided to purchase a product or service, the deal is typically done using a credit card. As the cardholder is not physically present, there are a number of risks (such as the use of stolen credit card numbers) that are traditionally controlled by such rules as 'goods bought by mail or telephone order must be dispatched to a cardholder address verified with the credit card issuer'. However, transactions may now involve the delivery of intangible goods to an email address, and this may create new risks.

Thirdly, it is possible for customers to make perfect copies of digital goods such as music and software, and pass these on to their friends. Pirates may even mass manufacture them and sell them at a discount. How can copying be controlled, given the ope nature of the medium and the difficulty of making consumer electronic devices really tamper-resistant?

The second and third problems have been studied extensively. The industry appears to have settled on SSL/TLS as a de facto standard for online credit card payments; meanwhile, there are many moves a foot to control copying, ranging from the launch of new digital audio formats to an upgraded version of DVD. However, the first problem – secure publication – has so far received little attention.

Yet secure publication is important. Although possible embarrassment from a hacked web site might be the first risk that springs to mind, there are much worse financial exposures (e.g., if a stock manipulator can pass off bogus news about a non-existent merger or acquisition). There are also serious safety hazards: for example, doctors are starting to rely on online reference sources for data on drug doses and reactions.

In this paper we describe a security policy model that seeks to deal in a quite general way with the issue of secure publication, whether of books, music, software, public key certificates, videos, drug formularies, aircraft maintenance manuals, academic papers or business records. We describe a set of mechanisms to support authentication and integrity in a quite general way in such environments, with a particular emphasis on audit and accountability. We sketch the beginnings of an authentication logic to help us analyse whether a particular protocol is sound. Finally, we describe a prototype implementation, which we call Jikzi, after the first ever book published using a movable type printing press; this is a Buddhist text printed in Korea in 1377, some 63 years before Gutenberg.

2 Previous Work

The direct precursor of Jikzi was the Wax system, developed to secure the online publication of medical information such as drug data [?]. The main problems we encountered were the difficulty of securing long-lived data such as books using relatively short-lived objects such as public keys, and the performance penalty imposed by trying to verify the signature on a whole book when the reader simply wished to consult a single section of it. Wax solved these problems by associating with each publisher a tree of hashes: the leaves were hashes of the individual sections, and the nodes progressed up through chapters and books until the root, which protects a publisher's whole catalogue, is authenticated by other means (typically a one-time signature). Thus when a user opens a book at the entry corresponding to a particular drug, the entry can be verified quickly without having to check a signature on the entire book. The lesson learned from Wax was that it is a sound design principle in publishing to use hash trees and to minimise the amount of signature – perhaps to a single signature on each new version of a publisher's catalogue.

The Wax design has had some success in the UK medical informatics field. But it uses a proprietary hypertext format, so the next step was to generalise it to work with html. The result was the ERL, or eternal resource locator [?], which can be thought of as a URL plus a hash of what you expect to find there. Having developed this, we realised that it was very powerful and useful: the writer of a web page or other document can authenticate any digital object by simply including its ERL, and this enables the construction of webs of trust of the kind familiar from PGP but applying quite generally to objects rather than just to the names of principals. Many of the problems associated with public key infrastructures simply go away; one ends up managing a few root keys and doing a lot of version control.

It has now become clear that we need to support XML as this is the emerging framework for commercial markup languages [?]. XML is well positioned between SGML and HTML; it is easier to use than SGML, yet and provides more degrees of freedom in extension than HTML. Major software vendors are committed to supporting it, including both major players in the browser market, who already support XML processing partially and are expected to support it fully in the near future.

The goal of the Jikzi project is thus to develop general XML-based mechanisms for embedding security properties in hypertext documents, and to understand the underlying computer science issues such as resilience, scalability and verification.

This brings us to other work in the same general field. Two particular projects bear mention: the Secure Document Markup Language (SDML) developed by Kravitz at IBM [?] which defines the tags necessary to implement electronic cheques ('payee', 'amount', 'currency', 'signature', 'certificate') and the World Wide Web Consortium's DSig project [?] to implement digital signature technology in content marking and rating schemes. While both of these projects are useful and instructive, we need something more general; we want to deal not just with cheques but with all other kinds of bills of exchange (such as bills of lading) and we want to be able to deal with inheritance properties (a bank cheque is a kind of cheque, and a cheque is a kind of bill of exchange).

A general security system should also be able to deal with applications such as military messaging: governments are likely to remain wedded to the idea of having paragraph-level security labels (i.e., a document is mixture of paragraphs at different levels such as unclassified and secret), and XML appears to be the only serious way in which the relevant document management routines can be implemented without rewriting a huge amount of software.

We also ought to be able to deal with publishing in the most general sense, which means dealing with issues of replication, persistence and control. Previous work in this area includes the Distributed Object-based Document Architecture (DODA) [?] which is aimed at computer supported cooperative work, such as software development; it is somewhat like a secure RCS in which 'folios' of documents are bundled together and managed by hash and signature mechanisms according to a policy such as 'a software upgrade can only be issued when both a testing manager and a customer manager sign it', and such policies can be enforced using mechanisms such as threshold signature.

3 A New Conceptual Model

Academic security research has tended to be 90% on confidentiality, 9% on authentication/integrity and perhaps 1% on availability. Yet commercial information systems departments have the opposite priorities. Availability of service is tops, with perhaps a third of the whole budget going on hot spare backup sites, network redundancy, and so on; integrity is next with several percent of the budget going on consistency checking mechanisms, internal audit and the like; but confidentiality is not a high priority at all [?]. In the world of publishing, this is especially so. So what would a security policy model for publishing look like?

At its simplest, a publisher is a principal who from time to time creates and makes world-readable a document such as a book, and also from time to time creates and makes world-readable a catalogue or index of such documents.

Stripping this to the bare essentials, we posit a universe in which each user can append data to a single file, which all other users can read. Deletion of data – even in one's own file – is not allowed. Thus we exclude consideration of confidentiality for the time being; the integrity of all strings once written is guaranteed; the authenticity of the author of a string is obvious; and we make digital signatures trivial (I can sign anything by simply asserting it in my file). The immutability of data once written means that a user can countersign a document by reference, e.g. 'I hereby concur with document number 382 in Bob's file'. We will ignore complex data structures such as indexes, as the mechanisms whereby they can be created from flat append-only files are well known.

We will now discuss append-only file systems and their implications for modeling security policy.

3.1 Append-only file systems

Append-only file systems are rarely considered in the research literature, but are remarkably common in real life. Banks must keep records for a period of time set by law (6 years in the UK), and much the same applies to medical records, though the period for which a record must be kept depends on the condition to which it relates – cancer records, for example, are kept for the patient's lifetime [?]. Business records in general are kept for the period during which transactions can be challenged by customers or audited by tax inspectors.

Publishing is perhaps the oldest application of append-only file systems. We do not expect that the archives of a newspaper or journal will ever have their content changed (except in the shrinking number of totalitarian states); if an issue of a publication is found to be defamatory or in breach of copyright, the appropriate legal remedy is financial compensation rather than retrospective modification. (In fact, the design of the Eternity Service was motivated by concern that once a book is no longer scattered round the world in tens of thousands of copies in libraries and private collections, but held electronically on three or four servers, then the retrospective modification or deletion of published material might become possible, with drastic long-term social and political consequences.)

Similarly, in commerce, the need for finality of transactions dictates that if a bank transaction is processed in error, it is not retrospectively deleted but rather a correcting transaction is passed through the system. (In the days when bank ledgers were kept on paper, they were kept in ink using bound books.)

There are many mechanisms which can be used to implement an append-only file store, ranging from CD-WORM drives through RCS to mainframe system managed storage products. In some applications, such as retail banking, the volume of transactions is large and thus the disk capacity required is fairly predictable; in others, such as medical records, the quantity of data which a doctor can type into his PC is negligible compared with the size of even the smallest hard disk; in others, there might be some risk of denial-of-service attacks involving resource exhaustion, and so one might either use disk quotas or insist on micropayments for disk writes.

The academic literature on append-only file systems, however, appears to be rather sparse. The earliest mention of which we are aware in the is the use of a public bulletin board by Benaloh to ensure the serialisation of pseudonymous votes cast in a digital election [?]. More generally, log structured file systems improve the performance of some systems by ensuring faster writes at the expense of longer read times [?, ?]. In systems in which erasure is expensive, such as flash memory, they allow an even distribution of writes in the media [?]. We might also mention the Eternity Service [?], a distributed file store in which techniques of fragmentation, redundancy, scattering and anonymity are used to provide a file store with the highest possible level of protection against service denial attacks: in effect, an append-only file store which is highly resistant to deletion attempts.

3.2 Example application

The reason we choose to use an append-only file system as our fundamental primitive is that it greatly simplifies many applications involving availability, integrity and authentication properties. By so doing, it forces us to think clearly about what optimisations we are making, and why, when we depart from it.

A good example is the B-money system proposed by Wei Dai [?]. It assumes that each user maintains a separate account of how much money everyone in the system owns, and in order to make a payment, one simply broadcasts a signed message saying 'I, Alice, hereby pay the sum of X to Bob'. B-money assumes a synchronous, unjammable broadcast channel, but can clearly be implemented in our model by having the principals append the transaction to their file. Secure serialisation is required but can be implemented in the obvious way using Lamport time [?] (but see below).

B-money has some interesting properties. For example, if someone computes the account balances of other parties incorrectly, it is himself that he puts at risk; he might accept a payment where the payer had insufficient funds, and then be embarrassed when one of his own payments was rejected by a more careful principal on the grounds that he in turn could not cover the payment.

This is a surprisingly apt model of a payment system: in the banking world, journal files are used to accumulate transactions from cash machines, cheque sorters, teller stations etc and these are applied overnight to the account master file in order to update it. The journals fill the same role as the transactions in b-money, and the overnight run is an optimisation which simplifies processing. To the extent that a bank's functions are mechanical (rather than to do with credit assessment and risk management), it might be replaced by a b-money system plus a software agent that performs this overnight update.

3.3 Significance

A security policy model, such as Bell-LaPadula [?] or Clark-Wilson [?], is a simplified way of looking at system security properties. It usually corresponds only partially to the reality of fielded systems, but is instructive in that the optimisations in these systems become clear for what they are.

The Jikzi model of the universe, as a set of append-only files, fulfills exactly this function. If the persistence and integrity of other principals' records could be relied on, then the design of business record systems would be much simpler; as it is, a prudent business will not trust all its counterparties to keep full records and make them available timely.

Thus instead of trusting that Bob's file is truly append-only and countersigning by reference ('I hereby concur with document number 382 in Bob's file'), a businessman in the real world will countersign by value ('I hereby concur with Bob's statement X and cite his signature S') or hash ('I hereby concur with Bob's statement whose SHA-1 is X and cite his signature S on it'). Whether one retains a copy of the message, or just the hash, will depend on whether one expects to have to produce the message at some time in the future, or whether the production would only be of value to another party (in which case there are trust models in which one still wants to retain the hash in order to have a complete list of the documents one had ever signed with a given key [**?**]).

It may be less clear that what we have created with this model is not just a (relatively) trivial way of modeling what happens in an ideal electronic publishing house, but also a principled way of looking at business records. Previously, the main security policy model proposed for commercial computer systems – Clark-Wilson – focussed entirely on the transactional aspects (e.g., how to ensure that it took two employees working together to raise a cheque, and that an indelible log of their actions was kept).

The record keeping aspects have so far escaped, perhaps because until very recently all business records were kept in paper form (even if the working copies were electronic) and so the issue of managing evidence in electronic form was just not considered. However, as electronic commerce spreads, this issue will become ever more important, and the Jikzi model provides a way of understanding it – that I must keep a copy of that part of the universe on which I rely. (There are some aspects of the universe that it might not be practical to keep; if I have an old VisiCalc file, do I need to keep an Apple II computer in working order so that I can browse it? However, our model can at least help people identify and assess such difficult dependencies.)

Finally, work on the legal validity of electronic signatures has so far focussed on the signature itself, type approval of the signature creation device, licensing and insurance of CAs, and so on. However, this is only the tip of the iceberg; context, and in particular the relationship between documents, is vital in real court cases. Did the hospital's web site list the side effects of drug X when the plaintiff visited it? Did it point to version 1.27 or 1.28 of MegaDrugCorp's fact sheet at the time? In general, who knew what when, which document was signed first, and how does the chain of evidence hang together?

It is for reasons like this that a prudent organisation, seeking to archive its business records, will not just timestamp and store html pages; we need some means of protecting – and auditing the use of – links and other dynamic aspects of the content. Here, the Jikzi approach comes into its own.

4 Towards an Authentication Logic

As this is a protocols workshop, it is natural for the reader to ask what sort of authentication protocols are natural in the Jikzi model. In fact, if the append-only model takes care of all integrity and authenticity concerns, do we need any protocols at all?

The answer, perhaps surprisingly, is yes. Time is not included in the basic security model, and so the fact of one event's having happened after another must be established explicitly. This is also true in conventional publishing, where dependence is typically established by citation. But even this is not straightforward; there are many cases of papers in a learned journal citing each other, and in general, where something predictable is going to be written, it can be cited in advance of its publication.

Thus in order to be sure that someone has read a message which I have written, I should include in it something unpredictable, such as a random number. In this way, incontrovertible serialisation can in principle be established, and constructs such as Lamport time can follow.

Suppose that Alice wants to ensure that after she makes some statement X, say, Bob reads it before responding Y. Our first attempt at such a protocol was:

$A \longrightarrow B : B, X, N_B$
$B \longrightarrow A : A, Y, h(X, N_B)$

Here, the element '$A \longrightarrow B : B, X, N_B$' means that A writes, for B to read, the values B, X and N_B in succession.

We looked at a number of verification logics to see which of them might be adapted to our purpose. The primitive we need seems to be something like:

B **said** Y **having seen** X

and this should be derived from a rule somewhat like the following:

B **said** (X, Y), X **unpredictable** *implies* B **said** Y **having seen** X

The elaboration of this into a formal verification system is work in progress; we report it in the tradition of the Cambridge protocols workshop. There are two interesting things that we have noticed so far. Firstly, in Kailar's logic of accountability [**?**], we cannot distinguish between different instances of a receipt; thus the verification of the IBS protocol is incomplete, as IBS [**?**] is silent on whether receipts have serial numbers (in fairness, its successors did have; the point is that the verification might usefully have dealt with these). Secondly, in the BAN logic [**?**], a nonce is a nonce: any distinction between a random challenge, a serial number and a timestamp is made in the protocol idealisation stage. Yet here, uniqueness and unpredictability are definitely different. For example, we probably want to have a rule saying something like:

B **said** Y **having seen** X, X **unpredictable** *implies* Y **came after** X

But it is not quite that simple. For example, if Alice quotes a message of mine which I know to have been unpredictable because it contained a random number which I generated, then I know that she read it; however, Bob might suspect me of having colluded and told her the random number. So unpredictability is in the eye of the beholder, and we may have to refine our rule to something like:

B **said** Y **having seen** X, A **believes** X **unpredictable** *implies*
A **believes** Y **came after** X

We would not claim that a short discussion of the rules a suitable logic might contain, amounts to the definition of a logic (that is for future work). However, even the discussion thus far leads us directly to our first ‘scalp’: the above draft protocol is wrong. It should be

$B \longrightarrow A : A, h(X, N_B), Y$

This was not at all obvious at the outset, and it is not clear how the mistake could have been found with an existing tool.

A final point, which we discussed at the workshop presentation but do not have the space to elaborate here, is that when one combines the Jikzi policy model with software agents (such as the agent we mentioned as a possible implementation of a bank in the b-money scenario above) then the result is a model that is somewhat similar to Clark-Wilson but somewhat more precise. We plan to develop this theme in a later paper.

5 The Jikzi Prototype

As mentioned above, one purpose of the Jikzi project was to extend the previous prototypes of securing documents in HTML and SGML such as ERL and SDML, using XML. XML enables the automation of document handling and interpretation using DTD (document type definitions), CSS (cascading style sheets) [**?**] and XSL (extensible stylesheet language) [**?**].

Jikzi uses its own markup language based on XML to support basic security features such as hashing and signature. The fastest way to explain it is probably by example, so we now describe how to write a electronic checque. First, we define a set of entities which will be used throughout our DTD files: `stdDef.dtd` contains the basic entity definitions. It also includes declarations for DTD files.

```
<!-- stdDef.dtd: DTD for entities -->
<!ENTITY % nameList "foreName CDATA #REQUIRED
    surName CDATA #REQUIRED
    initial CDATA #IMPLIED">
<!ENTITY % dateList "year CDATA #REQUIRED
    month CDATA #REQUIRED
    day CDATA #REQUIRED
    hour CDATA #IMPLIED
    minute CDATA #IMPLIED
    second CDATA #IMPLIED">
<!ENTITY % algoList "algoName CDATA #REQUIRED
    version CDATA #IMPLIED">
<!ENTITY % orgList "orgName CDATA #REQUIRED
    dept CDATA #IMPLIED
    section CDATA #IMPLIED">
<!ELEMENT dtd (dtdInfo)+>
<!ELEMENT dtdInfo EMPTY>
<!ATTLIST dtdInfo name CDATA #REQUIRED
    version CDATA #REQUIRED
```

```
    URL CDATA #IMPLIED
    author CDATA #IMPLIED>
<!-- end of stdDef.dtd -->
```

signList.dtd is used for signing a document; it contains signature information, plus the public key information needed to check the signature.

```
<!-- signList.dtd: DTD for signing docs -->
<!ENTITY % stdDef SYSTEM "http://www/~jhl21/dtds/stdDef.dtd">
%stdDef;
<!ELEMENT signList (sign)+>
<!ELEMENT sign (signInfo, pKeyInfo, signature)>
<!ELEMENT signInfo (signer, signAlgo, url?, parent?)>
<!ELEMENT signer EMPTY>
<!ATTLIST signer %nameList; orgName CDATA #IMPLIED
    signerId CDATA #IMPLIED>
<!ELEMENT signAlgo EMPTY>
<!ATTLIST signAlgo %algoList;>
<!ELEMENT url (#PCDATA)>
<!ELEMENT parent (#PCDATA)>
<!ELEMENT pKeyInfo (pKeyVersion, cert, pKey)>
<!ELEMENT pKeyVersion (#PCDATA)>
<!ELEMENT cert EMPTY>
<!ATTLIST certIssuer CDATA #REQUIRED
    certSerial CDATA #REQUIRED
    certUrl CDATA #REQUIRED
    revokeUrl CDATA #REQUIRED>
<!ELEMENT pKey (#PCDATA)>
<!ELEMENT signature (#PCDATA)>
<!-- end of signList.dtd -->
```

eCheque.dtd is a simple electronic cheque definition, and uses signList.dtd. It includes the cheque number, account number, payee, payer(s), payment amount, issue date, and timestamp. (A currency code can be specified with the payment amount.)

```
<!-- eCheque.dtd: DTD for electronic cheques -->
<!ENTITY % signList SYSTEM "http://www/~jhl21/dtds/signList.dtd">
%signList;
<!ELEMENT eCheque (dtd, chequeBody, signList)>
<!ELEMENT chequeBody (chequeId, account,
    payee, payment, issueDate, notLater?, timestamp, signInfo+)>
<!ELEMENT chequeId (#PCDATA)>
<!ELEMENT account (#PCDATA)>
<!ELEMENT payer (#PCDATA)>
<!ELEMENT payee EMPTY>
<!ATTLIST payee %nameList; payeeId CDATA #IMPLIED>
<!ELEMENT payment EMPTY>
```

```
<!ATTLIST payment amount CDATA #REQUIRED
    currency CDATA #IMPLIED>
<!ELEMENT issueDate EMPTY>
<!ATTLIST issueDate %dateList;>
<!ELEMENT notLater EMPTY>
<!ATTLIST notLater %dateList;>
<!ELEMENT timestamp (#PCDATA)>
<!-- end of eCheque.dtd -->
```

Electronic cheques are thus, as in SDML, just like paper cheques. There are external dependencies, or course, such as the certificates (which need to link the account number to the signer's distinguished name). In a real implementation, the dependencies are likely to get so complex that formal tools may be advisable in order to check for properties such as completeness and consistency.

Anyway, an actual electronic cheque looks like this:

```
<!-- corpCheque.xml -->
<?xml version="1.0"?>
<!DOCTYPE eCheque SYSTEM "eCheque.dtd">

<eCheque>
<dtd>
   <dtdInfo name="stdDef.dtd" version="1.0"/>
   <dtdInfo name="signList.dtd" version="1.0"/>
   <dtdInfo name="eCheque.dtd" version="1.0"/>
</dtd>
<chequeBody>
    <chequeId>00883627</chequeId>
    <account>23-45-67 1234567</account>
    <payer>University of Cambridge</payer>
    <payee foreName="William" surName="Hopkinson" initial="F"/>
    <payment amount="19.95" currency="UKP"/>
    <issueDate year="1999" month="01" day="15"/>
    <notLater year="1999" month="06" day="30"/>
    <timestamp>872043082393</timestamp>
    <signInfo>
        <signer foreName="John" surName="Smith" initial="M"/>
        <signAlgo algoName="PGP-RSA" version="5.5"/>
    </signInfo>
    <signInfo>
        <signer foreName="Edward" surName="Thompson" initial="J"/>
        <signAlgo algoName="PGP-DSS" version="5.5"/>
    </signInfo>
</chequeBody>
<signList>
<sign>
<signInfo>
    <signer foreName="John" surName="Smith" initial="M"/>
```

```xml
    <signAlgo algoName="PGP-RSA" version="5.5"/>
</signInfo>
<pKeyInfo>
    <pKeyVersion>1.0</pKeyVersion>
    <cert certIssuer="Cambridge Certificate Agency"
        certSerial="1234567890"
        certUrl="http://www.cca.com/certs/1234567890"
        revokeUrl="http://www.cca.com/revoke?sn=1234567890">
    <pKey>
    lQMFEDWhboWuyrPDhRvRXQEBkp4D/ivwpsci5MJQXUA
    bcPOUQquOgzMpp7W5KXP1Cit9EyqaPtet+1nkaoRXYv
    FQIB/eBjkcvNaA02w/mvHQRQYiAzz6kdPSn/rt9THkX
    LAOsOekv
    =1zy8
    <pKey>
</pKeyInfo>
<signature>
CZ/SDEjG6wt7V3uXWbZGV0pVg5LJg8j7bONjtdDuAHy
IyeYFI87qHE=
=OTeM
</signature>
</sign>
<sign>
<signInfo>
    <signer foreName="Edward" surName="Thompson" initial="J"/>
    <signAlgo algoName="PGP-DSS" version="5.5"/>
</signInfo>
<pKeyInfo>
    <pKeyVersion>1.0</pKeyVersion>
    <cert certIssuer="Cambridge Certificate Agency"
        certSerial="2345678901"
        certUrl="http://www.cca.com/certs/2345678901"
        revokeUrl="http://www.cca.com/revoke?sn=2345678901">
    <pKey>
    SHllb24gTGVlICgxMDIOKSA8Sm9uZy1IeWVvbi5MZWV
    bcPOUQquOgzMpp7W5KXP1Cit9EyqaPtet+1nkaoRXYv
    FQIB/eBjkcvNaA02w/mvHQRQYiAzz6kdPSn/rt9THkX
    L12s7ejk
    IEz1y
    </pKey>
</pKeyInfo>
<signature>
E0a57bT2+xWWds0Jh3wpIqV25B6+ExJA6xnAB3Az5hd
xAEALBQYiHd
n/rt9
</signature>
</sign>
</signList>
</eCheque>
<!-- end of corpCheque.xml -->
```

(In the above example, signatures are truncated to save space.)

6 Architecture

In the Jikzi prototype, all documents published can be protected from modification and deletion by making the underlying document storage append-only. However, if such mechanisms could be relied on, then – as in our abstract security policy model – there would be no need for digital signatures, hashes and other such modification defection mechanisms. What we see such mechanisms actually achieving is to build, on top of a less reliable set of platforms, a document store that is more reliable by virtue of the documents, and the links and other relationships between them, being protected using hashing and digital signature mechanisms.

Ideally, a user should be able to follow the dependencies automatically and identify all of the documents on which she relies to support her business records, or to provide a fully documented medical record. This might not just include (for example) a copy of her record at her GP's surgery, but also copies of hospital records to which a discharge summary in the GP record referred, but also copies of the relevant entries from drug formularies or treatment protocols on which particular treatment decisions were based.

This is of course an ambitious program, and prototype currently just provides us with the means to play with security markup languages, DTDs and cascading style sheets, so that we can begin to explore this space.

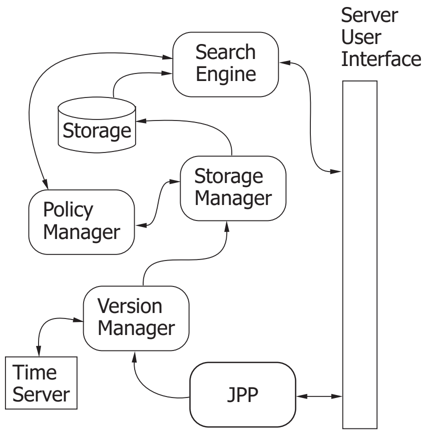

Figure 1. Architecture of the Jikzi system

The Jikzi server consists of the following blocks: Server User Interface, Jikzi PreProcessor, Version Manager, Storage Manager, Policy Manager, Search Engine, and storage. The Server User Interface has a web interface format so that users can access it with their browsers; pages for user document submission are are written in HTML/XML; and the documentation for the Jikzi service is written in XML. At this stage, a limited number of browsers support XML features and we believe that support of XML will expand.

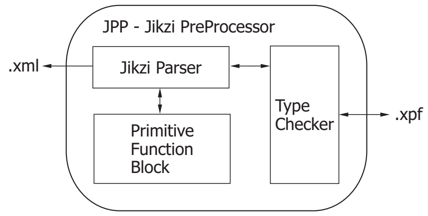

Figure 2. Jikzi PreProcessor

The Jikzi PreProcessor (JPP) consists of three parts: the Type Checker, the Jikzi Parser, and the Primitive Function Block. The JPP's input is in XML format and it is tested, manipulated, and then stored in an internal format called XML Primitive Format (XPF). The Type Checker verifies that submitted documents are well-formed. The Jikzi Parser catches Jikzi specific tags and embeds the function they need into the document. The Primitive Function Block is a library of functions used in Jikzi server which includes hashing, signatured, and verification algorithms.

The Version Manager acts like RCS; it does version control of all published files and keeps traces of their update history. Files are stored in a log-structured way for efficiency. It also lets users browse the history of published documents.

The Storage Manager is a front-end to the file system and enforces the access control policy laid down by the Policy Manager. The Policy Manager is a directory for policies and supports their registration and deregistration. It also interacts with the Search Engine. The Search Engine fetches documents requested by users on the basis of string matching. The Storage database, at least in our prototype, keeps all published documents in an append-only file.

The Time Server enables users to affix timestamps to publications. It can use either an external time service like the Network Time Protocol server [?] or an internal clock. At present, we use the latter.

This is still an alpha system and under development. Its purpose is to enable us to understand the issues involved with security markup languages, and the more input we can get, the better. The system is online at `<http://han.al.cl.cam.ac.uk>`, and if you want to be a tester, please contact the authors.

7 Conclusion

The three main aspects of electronic commerce that need to be protected against error and attack are publication, payment and copy control. So far the first of these has been largely ignored; in this paper we have tried to highlight the issues.

We have proposed a security policy model for publishing: a file store in which each user has append access to exactly one file, and all files are world readable. We have shown how some simple applications fit this model, and indicated its more general usefulness. We have sketched an authentication logic that enables us to reason about trust relationships. In the process, we have highlighted an ambiguity in the previous work, namely how one deals with different instances of a receipt.

We have also described a prototype system that we have developed to explore the use of authentication and integrity mechanisms in XML. This is an electronic publishing system that provides long term storage with support for XML and CSS, and features such as version management and timestamping.

Our work has some other interesting scientific aspects. It may help to elucidate our understanding of time and succession in the context of protocols and authentication logics, and the relationship between availability and authentication in scenarios where some views of a database are preserved and others lost. This may give us a better understanding of application level trust structures. Until now, models such as Clark-Wilson, Rampart [?] and Proactive Security [?] have tended to consider 'dual control' in the context of simple threshold schemes; but in real applications, one tends to find a much richer syntax of dependencies that are integrated with the underlying business model: it may take the assent of a number of principals with distinct roles to commit to a transaction, and a different set of principals to override it or recover from failure. This can give much greater resilience, but at a cost of more complexity – complexity familiar to commercial application developers, but which the research community has so far largely ignored.

The arrival of XML is bound to force application security syntax to be examined, and Jikzi provides the context in which this can be done.

8 Acknowledgement

The second author is supported by the EPSRC under grant number GR/L95809 on Resilient Security Mechanisms.