Security Policies

Ross Anderson <ross.anderson@cl.cam.ac.uk>
University of Cambridge Computer Laboratory

Frank Stajano <frank.stajano@cl.cam.ac.uk>, <fms@att.com>
University of Cambridge Computer Laboratory
AT&T Laboratories Cambridge

Jong-Hyeon Lee <jhlee@filonet.com>
Filonet Corporation

Abstract

A security policy is a high-level specification of the security properties that a given system should possess. It is a means for designers, domain experts and implementers to communicate with each other, and a blueprint that drives a project from design through implementation and validation.

We offer a survey of the most significant security policy models in the literature, showing how "security" may mean very different things in different contexts, and we review some of the mechanisms typically used to implement a given security policy.

Contents

1 What is a Security Policy?

Security engineering is about building systems to remain dependable in the face of malice as well as error and mischance. As a discipline, it focuses on the tools, processes and methods needed to design, implement and test complete systems, and to adapt existing systems as their environment evolves.

In most engineering disciplines, it is useful to clarify the requirements carefully before embarking on a project. Such a comment may sound so obvious as to border on the useless, but it is of special relevance to computer security. First, because it is all too often ignored [9]: diving straight into the design of crypto protocols is more fascinating for the technically minded. Second, because security is a holistic property — a quality of the system taken as a whole — which modular decomposition is not sufficient to guarantee. (We shall see in section 3.8.1 below that connecting secure components together does not necessarily yield a secure system.) It is thus important to understand clearly the security properties that a system should possess, and state them explicitly at the start of its development. As with other aspects of the specification, this will be useful at all stages of the project, from design and development through to testing, validation and maintenance.

A top down representation of the protection of a computer system might consist of the three layers shown in figure 1.

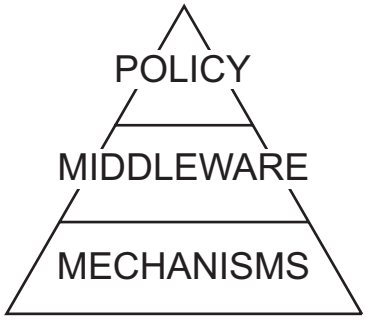

Figure 1: Layers of protection in a computer system

- At the highest level of abstraction, the whole system is represented by a concise and formalised set of goals and requirements: the **policy**.
- At the bottom level, the system is composed of **mechanisms** such as the computing hardware, the cryptographic primitives, tamper resistant enclosures and seals as well as procedural items such as biometric scanning of individuals (iris, fingerprint, voiceprint...) for purposes of authentication.
- Between those two extremes there will be some **middleware** that connects together the available mechanisms in order to build the system that conforms to the policy. This may include access control structures — whether or not enforced by the operating system — and cryptographic protocols.

The security policy is a set of high-level documents that state precisely what goals the protection mechanisms are to achieve. It is driven by our understanding of threats, and in turn drives our system design. Typical statements in a policy describe which subjects (e.g. users or processes) may access which objects (e.g. files or peripheral devices) and under which circumstances. It plays the same role in specifying the system's protection properties, and in evaluating whether they have been met, as the system specification does for general functionality. Indeed, a security policy may be part of a system specification, and like the specification its primary function is to communicate.

1.1 Definition

Many organisations use the phrase *security policy* to mean a collection of content-free statements. Here is a simple example:

Megacorp Inc security policy

1. This policy is approved by Management.
2. All staff shall obey this security policy.
3. Data shall be available only to those with a "need-to-know".
4. All breaches of this policy shall be reported at once to Security.

This sort of thing is common but is of little value to the engineer.

1. It dodges the central issue, namely 'Who determines "need-to-know" and how?'
2. It mixes statements at a number of different levels. Organizational approval of a policy should logically not be part of the policy itself (or the resulting self-reference makes it hard to express it formally).
3. The protection mechanism is implied rather than explicit: 'staff shall obey' — but what does this mean they actually have to do? Must the obedience be enforced by the system, or are users 'on their honour'?
4. It's unclear on how breaches are to be detected and on who has a duty to report them.

Because the term 'security policy' is so widely abused to mean a collection of platitudes, there are three more precise terms that have come into use to describe the specification of a system's protection requirements.

A **security policy model** is a succinct statement of the protection properties that a system, or generic type of system, must have. Its key points can typically be written down in a page or less. It is the document in which the protection goals of the system are agreed with an entire community, or with the top management of a customer. It may also be the basis of formal mathematical analysis.

A **security target** is a more detailed description of the protection mechanism that a specific implementation provides, and of how they relate to a list of control objectives (some but not all of which are typically derived from the policy model).

A **protection profile** is like a security target but expressed in an implementation-independent way to enable comparable evaluations across products and versions. This can involve the use of a semi-formal language, or at least of suitable security jargon. It is a requirement for products that are to be evaluated under the *Common Criteria* [61] (a framework used by many governments to facilitate security evaluations of defence information systems, and which we'll discuss below). The protection profile forms the basis for testing and evaluation of a product.

When we don't have to be so precise, we may use the phrase *security policy* to refer to any or all of the above. We will never use the term to refer to a collection of platitudes. We will also avoid a third meaning of the phrase – a list of specific configuration settings for some protection product. We will refer to that as *configuration management* in what follows.

1.2 Origins

Sometimes we are confronted with a completely new application and have to design a security policy model from scratch. More commonly, there already exists a model; we just have to choose the right one, and develop it into a protection profile and/or a security target. Neither of these tasks is easy. Indeed one of the main purposes of this chapter is to provide a number of security policy models, describe them in the context of real systems, and examine the engineering mechanisms (and associated constraints) that a security target can use to meet them. Let us then introduce, in chronological order, the three major waves of security policy models that have been presented in the open literature. We shall review them individually in greater detail in subsequent sections.

Historically, the concept of a security policy model came from the military sector. The first one to appear, **Bell-LaPadula** [14], was introduced

in 1973 in response to US Air Force concerns over the confidentiality of data in time-sharing mainframe systems. This simple yet influential model is based on restricting information flow between labelled clearance levels such as "Confidential" and "Top Secret". Its conceptual framework also forms the basis of other derived models such as **Biba** [17], which deals with integrity instead of confidentiality.

A second wave of policy models emerged in the 1980's from formalising well-established practices in the business sector. An abstraction of the double entry bookkeeping systems used in accounting and banking gave rise in 1987 to the **Clark-Wilson** security policy model [24]. Then Brewer and Nash, in 1989, introduced the **Chinese Wall** model [21] to represent the internal confidentiality constraints of a professional firm whose partners may be serving competing customers, and must avoid conflicts of interest.

A third wave came from the development of policy models for applications in various other fields — an activity that our group at Cambridge has pursued extensively in recent years. Case studies include the **BMA** (British Medical Association) security policy model [10], concerned with the confidentiality and accessibility of a patient's medical records; **Jikzi** [11] which describes the requirements of electronic publishing; and the **Resurrecting Duckling** [68], for secure transient association among, for example, wireless devices.

The lesson that can be drawn by observing such a wide spectrum of policy models is that security means radically different things in different applications. But whether we develop the system's security target using an established policy model or draw up a new model from scratch, a thorough understanding of the application environment and of established work patterns is essential, both to decide on a suitable model and to check that no threats have been overlooked. Consultation with domain experts is highly advisable. The wisdom provided by experience has few worthy substitutes and a careful study of the history of past attacks on similar systems is the best way to turn the ingenuity of yesterday's crooks to the advantage of today's designers [9].

2 The Bell-LaPadula Policy Model

By the early 1970's, people had realised that the protection offered by commercial operating systems was poor, and was not getting any better. As soon as one operating system bug was fixed, some other vulnerability would be discovered. Even unskilled users would discover loopholes and use them opportunistically.

A study by James Anderson led the US government to conclude that a secure system should do one or two things well, and that these protection properties should be enforced by mechanisms that were simple enough to verify and that would change only rarely [2]. It introduced the concept of a *reference monitor* – a component of the operating system that would mediate access control decisions and be small enough to be subject to analysis and tests, the completeness of which could be assured. In modern parlance, such components – together with their associated operating procedures – make up the *Trusted Computing Base* (TCB). More formally, the TCB is defined as the set of components (hardware, software, human, etc.) whose correct functioning is sufficient to ensure that the security policy is enforced — or, more vividly, whose failure could cause a breach

Figure 2: A classification hierarchy.

of the security policy. The goal was to make the security policy so simple that the TCB could be amenable to careful verification.

But what are these core security properties that should be enforced above all others?

2.1 Classifications and clearances

The Second World War, and the Cold War that followed, led NATO governments to move to a common protective marking scheme for labelling the sensitivity of documents. *Classifications* are labels such as *Unclassified*, *Confidential*, *Secret* and *Top Secret*, as in Figure 2. The original idea was that information whose compromise could cost lives was marked 'Secret' while information whose compromise could cost many lives was 'Top Secret'. Government employees have *clearances* depending on the care with which they've been vetted. The details change from time to time but a 'Secret' clearance may involve checking fingerprint files, while 'Top Secret' can also involve background checks for the previous five to fifteen years' employment [71].

The access control policy was simple: an official could read a document only if his clearance was at least as high as the document's classification. So an official cleared to 'Top Secret' could read a 'Secret' document, but not vice versa. The effect is that information may only flow upwards, from confidential to secret to top secret, but it may never flow downwards unless an authorized person (known as a *trusted subject*) takes a deliberate decision to declassify it.

There are also document handling rules; thus a 'Confidential' document might be kept in a locked filing cabinet in an ordinary government office, while higher levels may require safes of an approved type, guarded rooms with control over photocopiers, and so on. (The NSA security manual [58] gives a summary of the procedures used with 'Top Secret' intelligence data.)

The system rapidly became more complicated. The damage criteria for classifying documents were expanded from possible military consequences to economic harm and political embarrassment. The UK has an extra level, 'Restricted', between 'Unclassified' and 'Confidential'; the USA used to have this too but abolished it after the Freedom of Information Act was introduced. America now has two more specific markings: 'For Official Use only' (FOUO) refers to unclassified data that can't be released under FOIA, while 'Unclassified but Sensitive' includes FOUO plus material that might be released in response to a FOIA request. In the UK, 'Restricted' information is in practice shared freely, but marking low-grade government documents 'Restricted' allows journalists and others involved in leaks to be prosecuted. (Its other main practical effect is that when an unclassified US document is sent across the Atlantic, it automatically becomes 'Restricted' in the UK and then 'Confidential' when

shipped back to the USA. American military system people complain that the UK policy breaks the US classification scheme; Brits complain about an incompatible US refinement of the agreed system.)

There is also a system of codewords whereby information, especially at Secret and above, can be further restricted. For example, information which might reveal intelligence sources or methods — such as the identities of agents or decrypts of foreign government traffic — is typically classified 'Top Secret Special Compartmented Intelligence' or TS/SCI, which means that so-called *need to know* restrictions are imposed as well, with one or more codewords attached to a file. Some of the codewords relate to a particular military operation or intelligence source and are available only to a group of named users. To read a document, a user must have all the codewords that are attached to it. A classification level, plus a set of codewords, makes up a *security label* or (if there's at least one codeword) a *compartment*. Section 2.3 below offers a slightly more formal description, while a more detailed explanation can be found in [8].

Allowing upward only flow of information also models wiretapping. In the old days, tapping someone's telephone meant adding a physical tap to the wire. Nowadays, it's all done in the telephone exchange software and the effect is somewhat like making the target calls into conference calls with an extra participant. The usual security requirement is that the target of investigation should not know he is being wiretapped. What's more, a phone can be tapped by multiple principals at different levels of clearance. If the FBI is investigating a complaint that a local police force conducted an unlawful wiretap on a politician in order to blackmail him, they will also tap the line and should be able to see the police tap (if any) without the police detecting their presence.

Now that wiretaps are implemented as conference calls with a silent third party, care has to be taken to ensure that the extra charge for the conference call facility goes to the wiretapper, not to the target. (The addition of ever new features in switching software makes the invisible implementation of wiretapping ever more complex.) Thus wiretapping requires almost exactly the same information flow policy as does traditional classified data: High can see Low data, but Low can't tell whether High is reading any and if so what.

2.2 Automatic enforcement of information flow control

The next problem was how to enforce this information flow policy in a computer system. The seminal work here was the Bell-LaPadula (BLP) model of computer security, formulated in 1973 [14]. It is also known as **multilevel security**, and systems that implement it are often called *multilevel secure* or *MLS* systems. Their principal feature is that information can never flow downwards.

The Bell-LaPadula model enforces two properties:

- The *simple security property*: no process may read data at a higher level. This is also known as *no read up (NRU)*;
- The **-property*: no process may write data to a lower level. This is also known as *no write down (NWD)*.

The *-property was Bell and LaPadula's critical innovation. It was driven by the fear of attacks using malicious code. An uncleared user might write a Trojan and leave it around where a system administrator

cleared to 'Secret' might execute it; it could then copy itself into the 'Secret' part of the system and write secret data into unclassified objects that the attacker would later retrieve. It's also quite possible that an enemy agent could get a job at a commercial software house and embed some code in a product which would look for secret documents to copy. If it could then write them down to where its creator could read it, the security policy would have been violated. Information might also be leaked as a result of a bug, if applications could write down.

Vulnerabilities such as malicious and buggy code are assumed to be given. It is therefore necessary for the system to enforce the security policy independently of user actions (and by extension, of the actions taken by programs run by users). So we must prevent programs running at 'Secret' from writing to files at 'Unclassified', or more generally prevent any process at High from signalling to any object at Low. In general, when systems are built to enforce a security policy independently of user actions, they are described as having *mandatory access control*, as opposed to the *discretionary access control* in systems like Unix where users can take their own access decisions about their files. (We won't use these phrases much as they traditionally refer only to BLP-type policies and don't include many other policies whose rules are just as mandatory).

It should also be noted that another significant contribution of the work of Bell and LaPadula is not at the level of the security policy model itself but at the meta-level of talking about security policies in a formal way. Their presentation is based on a simple mathematical formalism that captures the security properties of interest and allows one to derive proofs about the security, or insecurity, of a given system.

2.3 Formalising the policy

Each subject and object in the system is assigned a *security label* or *protective marking* which consists of a number of *sub-markings* of which the most important is a *classification level* (e.g. Top Secret, Confidential etc) and a set of further sub-markings (*categories*) which means labels or compartments. A binary relation called "*dominates*" is then defined between any two security labels a and b in the following way.

$$\begin{gathered} \forall a, b \in labels : \\ a \ dominates \ b \\ \Updownarrow \\ level(a) \geq level(b) \ \wedge \ categories(a) \supseteq categories(b) \end{gathered}$$

This relation is a partial order, since it is antisymmetric and transitive. Given an appropriate set of security labels over which the two auxiliary functions $join()$ and $meet()$ can be defined, it forms a mathematical structure known as a lattice[1] [27], an example of which is shown in figure 3. Someone with a 'Top Secret' clearance isn't allowed to read a document marked $(Secret, \{Crypto\})$, despite it being at a lower level; he also needs

[1]The operators $join()$ and $meet()$ each take two elements from the set of labels and return another one, the least upper bound or the greatest lower bound respectively. Note that not all sets of labels give rise to a lattice under *dominates*: there may be sets of labels where one pair of elements does not have a least upper bound (or a greatest lower bound). For an example, remove the node $(Top\ Secret, \{Crypto, Foreign\})$ from figure 3.

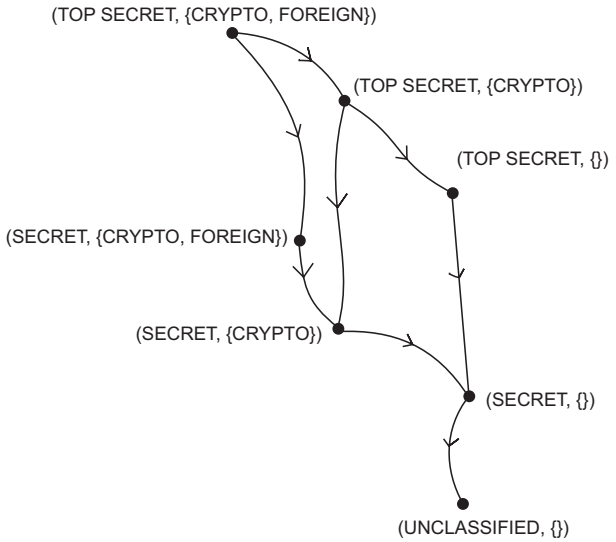

Figure 3: The "dominates" relation on a lattice of security labels.

a 'Crypto' clearance, which will place him at $(Top\ Secret, \{Crypto\})$, and this dominates the document's classification.

(Note that, to reduce clutter on such diagrams, it is customary to omit any arrows that can be deduced by transitivity.)

A predicate (i.e. a boolean-valued function) called "$allow()$", taking as arguments a subject, an object and an action, may then be defined in this framework. Stating a particular security policy is then equivalent to defining this function, which is in fact a complete formal specification for the behaviour of the reference monitor. The two rules of the BLP model can then be expressed as follows.

1. No read up (simple security property):

$$\forall s \in subjects, o \in objects :$$
$$allow(s, o, read) \Leftrightarrow label(s)\ dominates\ label(o)$$

2. No write down (star property):

$$\forall s \in subjects, o \in objects :$$
$$allow(s, o, write) \Leftrightarrow label(o)\ dominates\ label(s)$$

A state machine abstraction makes it relatively straightforward to verify claims about the protection provided by a design. Starting from a secure state, and performing only state transitions allowed by the rules of the chosen policy model, one is guaranteed to visit only secure states for the system. This is true independently of the particular policy, as long as the policy itself is not inconsistent. As we said, this idea of how to model a security policy formally was almost as important as the introduction of the BLP policy model itself.

This simple formal description omits some elaborations such as *trusted subjects* – principals who are allowed to declassify files. We'll discuss alternative formulations, and engineering issues, below.

2.4 Tranquility

The introduction of BLP caused some excitement: here was a straightforward security policy that appeared to be clear to the intuitive understanding yet still allowed people to prove theorems. But McLean [54] showed that the BLP rules were not in themselves enough. He introduced the conceptual construction of *System Z*, a system that suffered from blatant disclosure problems even though it officially complied with the letter of the BLP model. In System Z, a user can ask the system administrator to temporarily declassify any file from High to Low. So Low users can legitimately read any High file.

Bell's argument was that System Z cheats by doing something the model doesn't allow (changing labels isn't a valid operation on the state), and McLean's argument was that BLP didn't explicitly tell him so. The issue is dealt with by introducing a *tranquility property*. The strong tranquility property says that security labels never change during system operation, while the weak tranquility property says that labels never change in such a way as to violate a defined security policy.

The motivation for the weak property is that in a real system we often want to observe the principle of least privilege and start off a process at the uncleared level, even if the owner of the process were cleared to 'Top Secret'. If she then accesses a confidential email, her session is automatically upgraded to 'Confidential'; and in general, her process is upgraded each time it accesses data at a higher level (this is known as the *high water mark* principle). Such upgrades would not normally break a sensible security policy.

The practical implication of this is that a process acquires the security label or labels of every file that it reads, and these become the default label set of every file that it writes. So a process that has read files at 'Secret' and 'Crypto' will thereafter create files marked (at least) 'Secret Crypto'. This will include temporary copies made of other files. If it then reads a file at 'Top Secret Daffodil' then all files it creates after that will be labelled 'Top Secret Crypto Daffodil', and it will not be able to write to any temporary files at 'Secret Crypto'. The effect this has on applications is that most application software needs to be rewritten (or at least significantly modified) to run on MLS platforms.

Finally it's worth noting that even with this refinement, BLP still doesn't deal with the creation or destruction of subjects or objects, which is one of the hard problems of building a real MLS system.

2.5 Alternative formulations

System Z was one of several criticisms questioning the adequacy of the BLP model: this prompted research into other ways to describe multilevel secure systems and by now there are a number of competing models, some of which have been used to build real systems. We will now take a brief tour of the evolution of multilevel models, and without loss of generality we shall limit the discussion to two security levels, High and Low.

The first multilevel security policy was a version of high water mark written in 1967–8 for the **ADEPT-50**, a mandatory access control system developed for the IBM S/360 mainframe [75]. This used triples of level, compartment and group, with the groups being files, users, terminals and jobs. As programs (rather than processes) were subjects, it was vulnerable to Trojan horse compromises, and it was more complex than need be.

Nonetheless, it laid the foundation for BLP, and also led to the current IBM S/390 mainframe hardware security architecture.

The second was the **lattice model** that we mentioned above. A primitive version of this was incorporated into the the Pentagon's World Wide Military Command and Control System (WWMCCS) in the late 1960s, but this did not have the *-property. The realization that a fielded, critical, system handling Top Secret data was vulnerable to attack by Trojans caused some consternation [66]. Three improved lattice models were produced in the early 1970s: by Schell, Downey and Popek of the US Air Force in 1972 [67]; a Cambridge PhD thesis by Fenton, which managed labels using a matrix, in 1973 [36]; and by Walter, Ogden, Rounds, Bradshaw, Ames and Shumway of Case Western University who also worked out a lot of the problems with file and directory attributes [73, 74], which they fed to Bell and LaPadula [73, 74][2] Finally, the lattice model was systematized and popularized by Denning from 1976 [28].

Noninterference was introduced by Goguen and Meseguer in 1982 [41]. In a system with this property, High's actions have no effect on what Low can see. **Nondeducibility** is less restrictive and was introduced by Sutherland in 1986 [70]. Here the idea is to try to prove that Low cannot deduce anything with 100% certainty about High's input. Low users can see High actions, just not understand them; a more formal definition is that any legal string of high level inputs is compatible with every string of low level events. So for every trace Low can see, there is a similar trace that didn't involve High input. But different low-level event streams may require changes to high-level outputs or reordering of high-level/low-level event sequences.

The motive for nondeducibility is to find a model that can deal with applications such as a LAN on which there are machines at both Low and High, with the High machines encrypting their LAN traffic. (Quite a lot else is needed to do this right, from padding the High traffic with nulls so that Low users can't do traffic analysis, and even ensuring that the packets are the same size — see [65] for an early example of such a system.)

Nondeducibility has historical importance since it was the first nondeterministic version of Goguen and Messeguer's ideas. But it is hopelessly weak. There is nothing to stop Low making deductions about High input with 99% certainty. There are also many problems when we are trying to prove results about databases, and have to take into account any information that can be inferred from data structures (such as from partial views of data with redundancy) as well as considering the traces of executing programs.

Improved models include **Generalized Noninterference** and **Restrictiveness**. The former is the requirement that if one alters a high level input event in a legal sequence of system events, the resulting sequence can be made legal by, at most, altering subsequent high-level output events. The latter adds a further restriction on the part of the trace where the alteration of the high-level outputs can take place. This is needed for technical reasons to ensure that two systems satisfying the restrictiveness property can be composed into a third which also does. See [53] which explains these issues.

The **Harrison-Ruzzo-Ullman** model [43] tackles the problem of how

[2]Walter and his colleagues deserve more credit than history has given them. They had the main results first [73] but Bell and LaPadula had their work heavily promoted by the US Air Force. Fenton has also been largely ignored, not being an American.

to deal with the creation and deletion of files, an issue on which BLP is silent. It operates on access matrices and verifies whether there is a sequence of instructions that causes an access right to leak to somewhere it was initially not present. This is more expressive than BLP, but more complex and thus less tractable as an aid to verification.

Woodward proposed a **Compartmented Mode Workstation** (CMW) policy, which attempted to model the classification of information using floating labels, as opposed to the fixed labels associated with BLP [42, 78]. It was ultimately unsuccessful, because information labels tend to float up too far too fast (if the implementation is done correctly), or they float up more slowly (but don't block all the opportunities for unapproved information flow). However, CMW ideas have led to real products – albeit products that provide separation more than information sharing.

The **type enforcement** model, due to Boebert and Kain [20] and later extended by Badger and others [13], assigns each subject to a *domain* and each object to a *type*. There is a *domain definition table* (DDT) which acts as an access control matrix between domains and types. This is a natural model in the Unix setting as types can often be mapped to directory structures. It is more general than policies such as BLP, as it starts to deal with integrity as well as confidentiality concerns.

Finally, the policy model getting the most attention at present from researchers is **role-based access control** (RBAC), introduced by Ferraiolo and Kuhn [37]. This sets out to provide a more general framework for mandatory access control than BLP in which access decisions don't depend on users' names but on the functions which they are currently performing within the organisation. Transactions which may be performed by holders of a given role are specified, then mechanisms for granting membership of a role (including delegation). Roles, or groups, had for years been the mechanism used in practice in organisations such as banks to manage access control; the RBAC model starts to formalise this. It can deal with integrity issues as well as confidentiality, by allowing role membership (and thus access rights) to be revised when certain programs are invoked. Thus, for example, a process that calls untrusted software (which has, for example, been downloaded from the Net) might lose the role membership required to write to sensitive system files. We'll discuss this kind of engineering problem further below.

We won't go into the details of how to express all these properties formally. We will remark though that they differ in a number of important ways. Some are more expressive than others, and some are better at handling properties such as *composability* — whether a system built out of two components that are secure under some model is itself secure. We shall discuss this in section 3.8.1 below; for now, we will merely remark that two nondeducibility secure systems can compose into one that is not [52]. Even the more restrictive noninterference can be shown not to compose.

3 Examples of Multilevel Secure Systems

The enormous influence of BLP and its concept of multilevel security is perhaps best conveyed by a more detailed look at the variety of actual systems that have been built according to its principles.

Following some research products in the late 1970's (such as KSOS [16], a kernelised secure version of Unix), products that implemented multilevel

security policies started arriving in dribs and drabs in the early 1980's. By about 1988, a number of companies started implementing MLS versions of their operating systems. MLS concepts were extended to all sorts of products.

3.1 SCOMP

One of the most important products was the *secure communications processor* (SCOMP), a Honeywell derivative of Multics launched in 1983 [39]. This was a no-expense-spared implementation of what the U.S. Department of Defense believed it wanted: it had formally verified hardware and software, with a minimal kernel and four rings of protection (rather than Multics' seven) to keep things simple. Its operating system, STOP, used these rings to maintain up to 32 separate compartments, and to allow appropriate one-way information flows between them.

SCOMP was used in applications such as military *mail guards*. These are specialised firewalls that allowed mail to pass from Low to High but not vice versa [29]. (In general, a device that does this is known as a *data diode*.) SCOMP's successor, XTS-300, supports C2G, the Command and Control Guard. This is used in a Pentagon system whose function is to plan U.S. troop movements and associated logistics. Overall military plans are developed at a high classification level, and then distributed at the appropriate times as orders to lower levels for implementation. (The issue of how high information is deliberately downgraded raises a number of issues. In this case, the guard examines the content of each record before deciding whether to release it.)

SCOMP has had wide influence – for example, in the four rings of protection used in the Intel main processor line – but its most significant contribution to the security community was to serve as a model for the U.S. Trusted Computer Systems Evaluation Criteria (the *Orange Book*) [72]. This was the first systematic set of standards for secure computer systems, being introduced in 1985 and finally retired in December 2000. Although it has since been replaced by the Common Criteria, the Orange Book was enormously influential, and not just in America. Countries such as Britain, Germany, and Canada based their own national standards on it, and these national standards were finally subsumed into the Common Criteria [61].

The Orange Book allowed systems to be evaluated at a number of levels with A1 being the highest, and moving downwards through B3, B2, B1 and C2 to C1. SCOMP was the first system to be rated A1. It was also extensively documented in the open literature. Being first, and being fairly public, it set the standard for the next generation of military systems. This standard has rarely been met since; in fact, the XTS-300 is only evaluated to B3 (the formal proofs of correctness required for an A1 evaluation were dropped).

3.2 Blacker

Blacker was a series of encryption devices designed to incorporate MLS technology [15]. Previously, encryption devices were built with separate processors for the ciphertext, or *Black* end and the cleartext or *Red* end. There are various possible failures that can be prevented if one can coordinate the Red and Black processing. One can also provide greater operational flexibility as the device is not limited to separating two logical

networks, but can provide encryption and integrity assurance selectively, and interact in useful ways with routers. However, a high level of assurance is required that the Red data won't leak out via the Black. (For an actual example of such a leak, see [79].)

Blacker entered service in 1989, and the main lesson learned from it was the extreme difficulty of accommodating administrative traffic within a model of classification levels [76]. As late as 1994, it was the only communications security device with an A1 evaluation. So like SCOMP it influenced later systems. It was not widely used though, and its successor (the Motorola Network Encryption System) which is still in use, has only a B2 evaluation.

3.3 MLS Unix, CMWs and Trusted Windowing

Most of the available MLS systems are modified versions of Unix, and they started to appear in the late 1980's. An example is AT&T's System V/MLS [1]. This added security levels and labels, initially by using some of the bits in the group id record and later by using this to point to a more elaborate structure. This enabled MLS properties to be introduced with minimal changes to the system kernel. Other products of this kind included SecureWare (and its derivatives, such as SCO and HP VirtualVault), and Addamax.

Comparted Mode Workstations (CMWs) allow data at different levels to be viewed and modified at the same time by a human operator, and ensure that labels attached to the information are updated appropriately. The initial demand came from the intelligence community, whose analysts may have access to 'Top Secret' data, such as decrypts and agent reports, and produce reports at the 'Secret' level for users such as political leaders and officers in the field. As these reports are vulnerable to capture, they must not contain any information that would compromise intelligence sources and methods.

CMWs allow an analyst to view the 'Top Secret' data in one window, compose a report in another, and have mechanisms to prevent the accidental copying of the former into the latter (so cut-and-paste operations work from 'Secret' to 'Top Secret' but not vice versa). CMWs have proved useful in operations, logistics and drug enforcement as well [44].

For the engineering issues involved in doing mandatory access control in windowing systems, see [33, 34] which describe a prototype for Trusted X, a system implementing MLS but not information labelling. It runs one instance of X Windows per sensitivity level, and has a small amount of trusted code that allows users to cut and paste from a lower level to a higher one. For the specific architectural issues with Sun's CMW product, see [35].

3.4 The NRL Pump

It was soon realised that simple mail guards and crypto boxes were too restrictive, as many more internet services were developed besides mail. Traditional MLS mechanisms (such as blind write-ups and periodic read-downs) are inefficient for real-time services.

The US Naval Research Laboratory therefore developed the *Pump* – a one-way data transfer device using buffering and randomization to allow one-way information flow while limiting backward leakage [45, 47]. The attraction of this approach is that one can build MLS systems by using

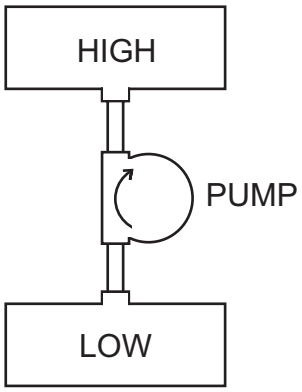

Figure 4: The pump.

pumps to connect separate systems at different security levels. As these systems don't process data at more than one level, they can be built from cheap commercial-off-the-shelf (COTS) components [46]. As the cost of hardware falls, this becomes the preferred option where it's possible.

The Australian government has developed a product called *Starlight*, which uses pump-type technology married with a keyboard switch to provide an MLS-type windowing system (albeit without any visible labels) using trusted hardware to connect the keyboard and mouse with High and Low systems [3]. There is no trusted software. It has been integrated with the NRL Pump [46]. A number of semi-commercial data diode products have also been introduced.

3.5 Logistics systems

Military stores, like government documents, can have different classification levels. Some signals intelligence equipment is 'Top Secret', while things like jet fuel and bootlaces are not; but even such simple commodities may become 'Secret' when their quantities or movements might leak information about tactical intentions. There are also some peculiarities: for example, an inertial navigation system classified 'Confidential' in the peacetime inventory might contain a laser gyro platform classified 'Secret'.

The systems needed to manage all this seem to be hard to build, as MLS logistics projects in both the USA and UK have been expensive disasters. In the UK, the Royal Air Force's Logistics Information Technology System (LITS) was a 10 year (1989–99), £500m project to provide a single stores management system for the RAF's 80 bases [57]. It was designed to operate on two levels: 'Restricted' for the jet fuel and boot polish, and 'Secret' for special stores such as nuclear bombs. It was initially implemented as two separate database systems connected by a pump to enforce the MLS property. The project became a classic tale of escalating costs driven by creeping requirements changes. One of these changes was the easing of classification rules with the end of the Cold War. As a result, it was found that almost all the 'Secret' information was now static (e.g., operating manuals for air-drop nuclear bombs that are now kept in strategic stockpiles rather than at airbases). In order to save money, the 'Secret' information is now kept on a CD and locked up in a safe.

Logistics systems often have application security features too. The classic example is that ordnance control systems alert users who are about to breach safety rules by putting explosives and detonators in the same truck or magazine [56].

3.6 Purple Penelope

In recent years, the government infosec community has been unable to resist user demands to run standard applications (such as MS Office) that are not available for multilevel secure platforms. One response is 'Purple Penelope'. This software, from a UK government agency, puts an MLS wrapper round a Windows NT workstation. This implements the high water mark version of BLP, displaying in the background the current security level of the device and upgrading it when necessary as more sensitive resources are read. It ensures that the resulting work product is labelled correctly.

Rather than preventing users from downgrading, as a classical BLP system might do, it allows them to assign any security label they like to their output. However, if this involves a downgrade, it requires the user to confirm the release of the data using a trusted path interface, thus ensuring no Trojan Horse or virus can release anything completely unnoticed. Of course, a really clever malicious program can piggy-back classified material on stuff that the user does wish to release, so there are other tricks to make that harder. There is also an audit trail to provide a record of all downgrades, so that errors and attacks (whether by users, or by malicious code) can be traced after the fact [63].

3.7 Future MLS systems

The MLS industry sees an opportunity in using its products as platforms for firewalls, Web servers and other systems that are likely to come under attack. Thanks to the considerable effort that has often gone into finding and removing security vulnerabilities in MLS platforms, they can give more assurance than commodity operating systems can that even if the firewall or Web server software is hacked, the underlying operating system is unlikely to be.

The usual idea is to use the MLS platform to separate trusted from untrusted networks, then introduce simple code to bypass the separation in a controlled way. In fact, one of the leading firewall vendors (TIS) was until recently a developer of MLS operating systems, while Secure Computing Corporation, Cyberguard and Hewlett-Packard have all offered MLS based firewall products. The long tradition of using MLS systems as pumps and mail guards means that firewall issues are relatively well understood in the MLS community. A typical design is described in [22].

3.8 What Goes Wrong

In computer security, as in most branches of engineering, we learn more from the systems that fail than from those that succeed. MLS systems have been an effective teacher in this regard; the large effort expended in building systems to follow a simple policy with a high level of assurance has led to the elucidation of many second- and third-order consequences of information flow controls.

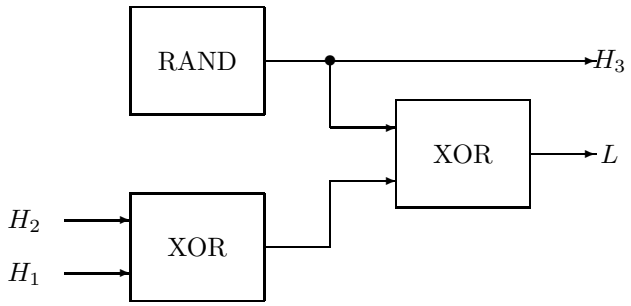

Figure 5: Insecure composition of secure systems with feedback.

3.8.1 Technical issues

One of the most intractable technical issues is *composability*. It is easy to design two systems that are each secure in themselves but which are completely insecure when connected together. For example, consider a simple device (figure 5) that accepts two High inputs H_1 and H_2; multiplexes them; encrypts them by xor'ing them with a one-time pad (i.e., a random generator); outputs the other copy of the pad on H_3; and outputs the ciphertext. Being encrypted with a cipher system giving perfect secrecy, this is considered to be Low (output L).

In isolation this device is provably secure. However, if feedback is permitted, then the output from H_3 can be fed back into H_2, with the result that the High input H_1 now appears at the Low output L.

This trivial example highlighted problems in connecting two devices of the same type, but things become significantly more difficult when dealing with heterogeneous systems. If the systems to be composed obey different policies, it is a hard problem in itself even to establish whether the policies can be made to be compatible at all! Lomas [50], for example, describes the difficulty of reconciling the conflicting security policies of different national branches of the same investment bank. In fact, establishing the conditions under which security policies compose is a long standing area of research.

There are many other technical problems. We will summarise them here briefly; for a fuller account, the reader should consult Anderson [8].

Covert channels arise when a high process can signal to a low process by affecting some shared resource. For example, it could position the disk head at the outside of the drive at time t_i to signal that the i-th bit in a Top Secret file was a 1, and position it at the inside to signal that the bit was a 0. A typical modern operating system has many such channels, which provide a means for a virus that has migrated up to 'High' to signal back down to 'Low'.

Polyinstantiation refers to the problem of maintaining data consistency when users at different clearance levels work with different versions of the data. Some systems conceal the existence of High data by inventing cover stories, and problems can arise if Low users then rely on these: it is easy to end up with multiple inconsistent copies of a database.

Aggregation refers to the fact that a collection of Low facts may enable an attacker to deduce a High one. For example, we might be happy to declassify any single satellite photo, but declassifying the whole collection would reveal our surveillance capability and the history of our intelligence priorities.

Overclassification is common. Because processes are automatically upgraded as they see new labels, the files they use have to be too. New files default to the highest label belonging to any possible input. The result of all this is a chronic tendency for objects to migrate towards the highest level of classification in the system.

Downgrading is a huge problem. An intelligence analyst might need to take a satellite photo classified at TS/SCI, and paste it into a'Secret' assessment for field commanders. This contravenes the BLP model, and so has to be handled by a trusted subject (that is, trusted code). But often the most difficult parts of the problem have to be solved by this code, and so the MLS mechanisms add little. They can provide very high quality data separation, but the real problems are more likely to lie in the controlled sharing of data.

Application incompatibility is often the worst problem of all; in many cases it is the show-stopper. For example, a process that reads a High file and is upgraded will automatically lose the ability to write to a Low file, and many applications simply cannot cope with files suddenly vanishing. The knock-on effects can be widespread. For example, if an application uses a license server and is upgraded, then the license server must be too, so it vanishes from the ken of other copies of the application running at Low, whose users get locked out.

These technical problems are discussed at greater length in Anderson [8]. They are important not just to builders of multilevel secure systems but because variants of them surface again and again in other systems with mandatory access control policies.

3.8.2 Political and economic issues

The most telling argument against MLS systems is economic. They are built in small volumes, and often to high standards of physical robustness, using elaborate documentation, testing and other quality control measures driven by military purchasing bureaucracies. Administration tools and procedures are usually idiosyncratic, which adds to the cost; and many applications have to be rewritten to cope with the MLS functionality.

One must never lose sight of the human motivations that drive a system design, and the indirect costs that it imposes. Moynihan provides a critical study of the real purposes and huge costs of obsessive secrecy in US foreign and military affairs [55]. Following a Senate enquiry, he discovered that President Truman was never told of the Venona decrypts because the material was considered 'Army Property' — despite its being the main motive for the prosecution of Alger Hiss. As his book puts it, "Departments and agencies hoard information, and the government becomes a kind of market. Secrets become organizational assets, never to be shared save in exchange for another organization's assets." He reports, for example, that in 1996 the number of original classification authorities decreased by 959 to 4,420 (following post-Cold-War budget cuts) but the total of all classification actions reported increased by 62% to 5,789,625. Yet despite the huge increase in secrecy, the quality of intelligence made available to the political leadership appears to have degraded over time.

Effectiveness is undermined by inter-agency feuding and refusal to share information, and by the lack of effective external critique. So a case can be made that MLS systems, by making the classification process easier and controlled sharing harder, actually impair operational effectiveness.

4 The Biba Integrity Model

The usual formal definition of mandatory access control is that information flow restrictions are enforced independently of user actions. Although this is often taken to mean BLP, the majority of fielded systems that enforce such controls do so to protect integrity properties. The typical example comes from an electricity utility, where the main operational systems such as power dispatching and metering can feed information into the customer billing system, but not vice versa. Similar one-way information flows are found in banks, railroads, hospitals and a wide range of other commercial and government systems.

The first security policy model to deal with such integrity protection was due to Biba [17] and is often referred to as 'Bell-LaPadula upside down'. Its key observation is that confidentiality and integrity are in some sense dual concepts — confidentiality is a constraint on who can read a message, while integrity is a constraint on who may have written or altered it.

In BLP, information cannot flow down towards levels of lower confidentiality, since this would cause a leak. In Biba, conversely, information cannot flow up towards levels of higher integrity, or the "impure" data from the lower-integrity levels would contaminate the "pure" data held in the higher levels. This may be formulated in terms of a No Read Down and a No Write Up property that are the exact dual of the corresponding ones in BLP.

Further applications in which Biba is often applied (without the system builders being even aware of its existence) include the following.

- An electronic medical device such as an ECG may have two separate modes: calibration and use. The calibration data must be protected from being corrupted by normal users, who will therefore be able to read it but not write to it. When a normal user resets the device, it will lose its current user state (i.e., any patient data in memory) but the calibration will remain unchanged.
- In computer supported cooperative work, some of the authors may be very careful in noting all the precise details of their bibliographic citations from first hand references, while others may content themselves with less complete records, perhaps only cross-checked on the Web rather than on the actual articles. In such a case, the more meticulous authors will probably refrain from copying citations from their colleagues' files (No Read Down) and not grant their colleagues permission to modify their own bibliography files (No Write Up).

The duality between Biba and BLP means that the System Z objections apply here too, and the introduction of tranquility properties becomes necessary. The obvious interpretation is a *low water mark* policy in which the integrity label of an object defaults to the lowest label read by the process that created it.

An example of implementation is LOMAC, an extension to Linux with a low water mark policy [40]. This is designed to deal with the problem of

malicious code arriving over the Net. The system provides two levels — high and low integrity — with system files at High and the network at Low. As soon as a program (such as a demon) receives traffic from the network, it is automatically downgraded to Low. Thus even if the traffic contains an attack that succeeds in forking a root shell, this shell won't have the ability to write to the password file, for example, as a normal root shell would. As one might expect, a number of system tasks (such as logging) become tricky and require trusted code. However, these mechanisms still cannot stop a virus that has infected Low from replicating and sending copies of itself out over the network.

As mentioned above, integrity concerns can also be dealt with by the type enforcement and RBAC models. However, in their usual forms, they revise a principal's privilege when an object is invoked, while low watermark revises it when an object is read. The latter policy is more prudent where we are concerned with attacks exploiting code that is not formally invoked but simply read (examples include buffer overflow attacks conducted by 'data' read from the Internet, and 'documents' that actually contain macros — currently the most popular medium for virus propagation).

An interesting problem is that of combining the apparently contradictory requirements of Biba and BLP, as would be needed in a system for which both confidentiality and integrity were equally important goals. The trivial approach based on a single set of security labels for both confidentiality and integrity leads to an extremely restrictive system in which information cannot flow either up or down, but only sideways, among items at the same security level; this does comply with both policies simultaneously, but probably does not yield a very useful system.

A more intriguing solution is to assign different labels for confidentiality and integrity, and in particular to make high integrity correspond to low confidentiality and vice versa. Depending on the context, this may not be as absurd as it may sound at first. Consider for example that systems software needs extremely high integrity, but very low confidentiality; whereas this may reversed for data items such as user preferences. This approach has the advantage that both policy models end up dictating information flow in the same direction [1]. Researchers are now starting to build more complex models that accommodate both confidentiality and integrity to observe their interaction [48].

5 The Clark-Wilson Model

Most mandatory integrity policies used in real systems are somewhat more complex than Biba, and the most influential of them is Clark-Wilson (CW). This model distills a security policy out of the centuries-old practice of double-entry bookkeeping (arguably one of the most significant ideas in finance after the invention of money). Its main goal is to ensure the integrity of a bank's accounting system and to improve its robustness against insider fraud.

The idea behind double-entry bookkeeping is, like most hugely influential ideas, extremely simple. Each transaction is posted to two separate books, as a credit in one and a debit in the other. For example, when a firm is paid $100 by a creditor, the amount is entered as a debit in the accounts receivable (the firm is now owed $100 less) and as a credit in the cash account (the firm now has $100 more cash). At the end of the

day, the books should *balance*, that is, add up to zero; the assets and the liabilities should be equal. (If the firm has made some profit, then this is a liability the firm has to the shareholders.) In all but the smallest firms, the books will be kept by different clerks, and have to balance at the end of every month (at banks, every day). By suitable design of the ledger system, we can see to it that each shop, or branch, can be balanced separately. Thus most frauds will need the collusion of two or more members of staff; and this principle of split responsibility is complemented by audit.

Similar schemes had been in use since the Middle Ages, and had been fielded in computer systems since the 1960's; but a proper model of their security policy was only introduced in 1987, by Clark and Wilson [24]. In their model, some data items are constrained so that they can only be acted on by a certain set of transactions known as transformation procedures.

More formally, there are special procedures whereby data can be input — turned from an *unconstrained data item*, or UDI, into a *constrained data item*, or CDI; *integrity verification procedures* (IVP's) to check the validity of any CDI (e.g., that the books balance); and *transformation procedures* (TPs), which may be thought of in the banking case as transactions that preserve balance. In the general formulation, they maintain the integrity of CDIs; they also write enough information to an append-only CDI (the audit trail) for transactions to be reconstructed. Access control is by means of triples (subject, TP, CDI), which are so structured that a dual control policy is enforced. Here is the formulation found in [1].

1. The system will have an IVP for validating the integrity of any CDI.
2. Application of a TP to any CDI must maintain its integrity.
3. A CDI can only be changed by a TP.
4. Subjects can only initiate certain TPs on certain CDIs.
5. CW-triples must enforce an appropriate separation of duty policy on subjects.
6. Certain special TPs on UDIs can produce CDIs as output.
7. Each application of a TP must cause enough information to reconstruct it to be written to a special append-only CDI.
8. The system must authenticate subjects attempting to initiate a TP.
9. The system must let only special subjects (i.e., security officers) make changes to authorisation-related lists.

One of the historical merits of Clark and Wilson is that they introduced a style of security policy that was not a direct derivative of BLP. In particular, it involves the maintenance of application-level security state — firstly, in the audit log, and secondly to track the shared control mechanisms. These can be quite diverse. They can operate in parallel (as when two bank managers are needed to approve a transaction over a certain amount) or in series (as when different people in a company are responsible for raising an order, accepting delivery, paying an invoice and balancing a departmental budget). Although the details can be highly application specific, this new approach provided an overall framework for reasoning about such systems, and fuelled research into security policies that were not based on label-based classification.

Despite being very different from the rules of the BLP model, the Clark-Wilson rules still fall into the general pattern of first defining a

subset of the states of the system as "secure", and then defining transition rules that, when applied to secure states, are guaranteed to lead into further secure states, thus preserving the fundamental invariant of the system. Insofar as a security policy is an abstract description of the desired behaviour of the Trusted Computing Base, the above pattern captures fairly well the concept of "security policy" as we defined it in the introduction.

6 The Chinese Wall Model

The Chinese Wall security policy, introduced by Brewer and Nash in 1989 [21], models the constraints of a firm of professionals – such as computer consultants, advertising agents or investment bankers — whose partners need to avoid situations where *conflicts of interest* or *insider dealing* might become possible.

Suppose the firm consults for a variety of companies, for example three oil companies, four banks and two computer companies. Any partner consulting for a company of a given type, say "oil company", would face a conflict of interest if she were also to consult for any other company of that type. But nothing should stop her from simultaneously consulting for a company of another type, such as a bank. As long as the consultant has not yet interacted with companies of a given type, she is free to choose any company of that type for a new assignment. However, as soon as she consults for one of them, a closed 'Chinese Wall' is erected around her, with that company inside and all the other companies of the same type outside. So the consultant's personal Chinese Wall changes whenever she consults for a company of a new type.

The authors explicitly compare their policy with BLP and describe it using a similar formalism based on a simple security property and a star property. These two rules are somewhat more complicated than their BLP equivalents.

Given a data object o, for example the payroll file of Shell, $y(o)$ indicates the company to which o refers, namely Shell, and $x(o)$ denotes the type of company, in this case "oil company", which may also be seen as the set of companies among which there is a conflict of interests from the point of view of an analyst who accesses o.

The critical difference from BLP is that the Chinese Wall model needs to retain *state* in order to keep track of the objects (and therefore companies) with which analysts have been "contaminated". The state is kept in a two-dimensional matrix of Boolean values, N, indexed by subject and object: $N_{s,o}$ is true if and only if subject s has previously accessed object o.

The simple security property says that each subject can access objects from at most one company of any given type. In particular, subject s can access object o only if one of the two following circumstances is verified. Either s has never dealt with a company of type $x(o)$, i.e., there is no object p such that $N_{s,p}$ is true and $x(p) = x(o)$; or s is already committed to the specific company $y(o)$, i.e., for each object p such that $N_{s,p}$ is true and $x(p) = x(o)$ we also have that $y(p) = y(o)$.

This still leaves scope for information leaks through indirect routes. Analyst Alice might be consulting for oil company Shell and bank Citicorp, while analyst Bob might be consulting for Exxon and Citicorp. Nothing as yet prevents Alice from writing Shell-related financial information in

a Citicorp object that Bob might later read, thus causing a conflict with Bob's allegiance to Exxon.

The star property covers this case. Subject s is only allowed to write to object o if the simple property is satisfied and if, for every object p that s has previously read, either $y(p) = y(o)$ or p is a "sanitised" object. To sanitise an object o is to transform it in such a way that no conflict of interest will occur if the sanitised object is disclosed to companies belonging to $x(o)$. This may be achieved through a trusted subject applying appropriate de-identification or other data laundering mechanisms. Sanitised objects can be elegantly included in the model by introducing an artificial company type "sanitised" containing only one company. Since the cardinality of the type is 1, such objects may be accessed by all analysts without any conflict of interests.

The Chinese Wall model made a seminal contribution to the theory of access control. It also sparked a debate about the extent to which it is consistent with the MLS tranquility properties, and some work on the formal semantics of such systems (see, for example, Foley [38] on the relationship with non-interference).

There are also some interesting new questions about covert channels. For example, could an oil company find out whether a competitor that used the same consultancy firm was planning a bid for a third oil company, by asking which specialists were available for consultation and noticing that their number had dropped suddenly?

7 The BMA Policy

The healthcare sector offers another interesting scenario in which confidentiality requirements are paramount, but radically different from those in the military context. Medical privacy is a legal right in many countries, and frequently a subject of controversy. As the information systems in hospitals and medical practices are joined together by networks, potentially large numbers of people have access to personal health information, and this has led to some abuses. The problem is likely to get worse as genetic data become widely available. In Iceland a project to build a national medical database that will incorporate not just medical records but also genetic and genealogical data, so that inherited diseases can be tracked across generations, has caused an uproar [7, 6].

The protection of medical information is also a model for protecting personal information of other kinds, such as the information held on individual customers by banks, insurance companies and government agencies. In EU countries, citizens have rights to *data protection*. In broad terms, this means that they must be notified of how their personal data may be used, and in the case of especially sensitive data (affecting health, sexual behaviour and preferences, political and trade union activity and religious belief) they either must give consent to information sharing or have a right of veto. This raises the issue of how one can construct a security policy in which the access control decisions are taken not by a central authority (as in Bell-LaPadula) or by the system's users (as in discretionary access control) but by the data subjects.

In a 1996 project for which one of us was responsible, the British Medical Association developed a security policy for clinical information systems [10]. This model focuses on access control, patient privacy and confidentiality management. It has a horizontal structure of access control

rather than a vertical hierarchy as used in the BLP model, although its confidentiality properties bear some relation to BLP as applied to compartments.

The goals of the BMA security policy were to enforce the principle of patient consent, and to prevent too many people getting access to too large databases of identifiable records. It did not try to do anything new, but merely to codify existing best practice. It also sought to express other security features of medical record management such as safety and accountability.

The policy consists of nine principles:

1. (Access control) Each identifiable clinical record shall be marked with an access control list naming the people or groups of people who may read it and append data to it. The system shall prevent anyone not on the access control list from accessing the record in any way.
2. (Record opening) A clinician may open a record with herself and the patient on the access control list. Where a patient has been referred, she may open a record with herself, the patient and the referring clinician(s) on the access control list.
3. (Control) One of the clinicians on the access control list must be marked as being responsible. Only she may alter the access control list, and she may only add other health care professionals to it.
4. (Consent and notification) The responsible clinician must notify the patient of the names on his record's access control list when it is opened, of all subsequent additions, and whenever responsibility is transferred. His consent must also be obtained, except in emergency or in the case of statutory exemptions.
5. (Persistence) No-one shall have the ability to delete clinical information until the appropriate time period has expired.
6. (Attribution) All accesses to clinical records shall be marked on the record with the subject's name, as well as the date and time. An audit trail must also be kept of all deletions.
7. (Information flow) Information derived from record A may be appended to record B if and only if B's access control list is contained in A's.
8. (Aggregation control) There shall be effective measures to prevent the aggregation of personal health information. In particular, patients must receive special notification if any person whom it is proposed to add to their access control list already has access to personal health information on a large number of people.
9. (Trusted computing base) Computer systems that handle personal health information shall have a subsystem that enforces the above principles in an effective way. Its effectiveness shall be subject to evaluation by independent experts.

This policy is strictly more expressive than Bell-LaPadula (it contains a BLP-type information flow control mechanism in principle 7, but also contains state). A fuller discussion from the point of view of access control, aimed at a technical audience, can be found at [4].

The fundamental innovation of the BMA model is not at first sight obvious. Previous models had tried to produce a security policy for the

'electronic patient record' — a phrase that has come to mean the entirety of a person's health information, from conception through autopsy. This turned out to be an intractable problem, because of the different groups of people who had access to different subsets of the record. So the solution adopted by the BMA model was to define the record as the maximal set of health information about a person that shared the same access control list.

A system now deployed in a number of British hospitals, which works along these lines and broadly complies with the BMA policy, is described in [26].

8 Jikzi

In the last decade of the twentieth century the World Wide Web staked a plausible claim to be the most significant event in publishing since the invention of the printing press. Anyone can now access, from anywhere in the world, a vast multifaceted and continuously updated hypertextual network of documents.

The problem with this new medium, compared with the established world of paper-based publishing, is its ephemeral nature. There is no guarantee that the interesting and useful Web page we are consulting now will be there tomorrow — or, more subtly, that it won't have been edited to deny something that it today asserts. There are few guarantees about the identity of the author of a page and any text can be repudiated at any time by withdrawing or modifying it.

These properties make the new medium questionable for applications that need better guarantees of integrity, such as drugs databases, company business records and newspaper archives[3].

To address the requirements of secure publishing, we implemented a system based on the idea of retaining all versions of the relevant documents forever, without ever deleting any of them. We called it 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).

The security policy model of the Jikzi system is as follows. As a foundation, we assume the domain D of published documents to be partitioned into two sets: the controlled documents, CD, whose integrity and authenticity are guaranteed by the system, and the uncontrolled documents, UD, on which no constraints are placed. The policy proper is stated as six principles:

1. Neither deletion nor replacement is allowed within CD.
2. The creator of a document defines its revision access condition and only authorised principals with respect to the condition are allowed to revise it; all revisions in CD must be stored and browsable.
3. Authenticity validation procedures must be available for validating the authenticity of CD members.
4. Any action to CD members must maintain the authenticity of the document.
5. Authentication of a member of CD can be performed by any user.

[3]George Orwell's famous novel *1984* depicts a world in which even paper-based newspaper archives were retroactively changed so that history would suit the current government.

6. Transformation from UD to CD must be one-way and the principal who transformed a document becomes the creator of the document in CD.

The policy protects controlled documents by preventing any destructive modifications to them. If we need to modify a controlled document, we may produce a revised copy of it, but all the previous versions will stay archived for the lifetime of the system.

Principle 6 assumes a transformation from an uncontrolled document to its controlled version. The one-wayness of the transform means that a document, once controlled, cannot become uncontrolled, since Principle 1 does not allow deletions from CD. It is of course possible to take an uncontrolled copy of a controlled document, and from then on the life of the copy will no longer be controlled; but the previous history of the document in CD remains unchanged.

In the paper-based commercial world, write-once documents have existed for a long time (e.g. business ledgers). One of us is using the above policy to build an online data repository service.

Conversely, sometimes our data will not require persistence, but rather volatility. As we shall discuss in section 11.2, there are plausible scenarios in which we may wish all controlled documents to *disappear* after a fixed delay since their creation.

9 The Resurrecting Duckling

Authentication of principals in a distributed system is a well studied problem with established solutions. The traditional solutions, however, rely on the existence of an online server, either for the distribution of "tickets", as in Kerberos[4], or to check whether a public key has been revoked, as in the various public key infrastructures[5]. Where such a server is not available, a new strategy must be found.

The problem becomes apparent in the example of a universal remote control that needs to be configured so as to control a new DVD player that its owner just bought. We want the DVD player to obey this remote control but not any other, so as to prevent accidental (or malicious) activation by our neighbour's remote control. We also want to be able to rescind this association, so that we may resell or give away the player once a better model comes out; but this facility should be restricted, to prevent a thief who steals the player from using it. The goal may be summarised as "secure transient association", and the abstract problem has much wider applicability than just consumer electronics: one may conceive further instances as diverse as the temporary binding of an e-wallet to an ATM (nobody should be able to interfere while I am performing my transaction, but the ATM should be ready to bind to another customer's e-wallet as soon as I go away) or as the temporary binding of a hospital thermometer to a doctor's PDA.

[4]Kerberos [60] is an online authentication protocol based on Needham-Schroeder [59] and developed at MIT for the Athena project in the late 1980s. The general idea is as follows. Client A wishes to access resource server B, and therefore needs to authenticate itself to B as a valid user. A and B don't know each other, but they each know (i.e. share a secret with) the authentication server S. So A contacts S and receives from it a time-limited "ticket" that it can show to B to prove its identity. A variant of Kerberos is used in Windows 2000.

[5]See section 11.1.

A metaphor inspired by biology will help us describe the security policy model we developed to implement secure transient association [68].

As Konrad Lorenz beautifully narrates [51], a duckling emerging from its egg will recognise as its mother the first moving object it sees that makes a sound, regardless of what it looks like: this phenomenon is called imprinting. Similarly, our device (whose egg is the shrink-wrapped box that encloses it as it comes out of the factory) will recognise as its owner the first entity that sends it a secret key through an electrical contact. As soon as this 'imprint key' is received, the device is no longer a newborn and will stay faithful to its owner for the rest of its life. If several entities are present at the device's birth, then the first one that sends it a key becomes the owner: to use another biological metaphor, only the first sperm gets to fertilise the egg.

We can view the hardware of the device as the body, and the software (and particularly its state) as the soul. As long as the soul stays in the body, the duckling remains alive and bound to the same mother to which it was imprinted. But this bond is broken by death: thereupon, the soul dissolves and the body returns in its pre-birth state, with the resurrecting duckling ready for another imprinting that will start a new life with another soul. Death is the only event that returns a live device to the pre-birth state in which it will accept an imprinting. We call this process *reverse metempsychosis*. Metempsychosis refers to the transmigration of souls as proposed in a number of religions; our policy is the reverse of this as, rather than a single soul inhabiting a succession of bodies, we have a single body inhabited by a succession of souls.

With some devices, death can be designed to follow an identifiable transaction. A hospital thermometer can be designed to die (lose its memory of the previous key and patient) when returned to the bowl of disinfectant at the nursing station. With others, we can arrange a simple timeout, so that the duckling dies of old age. With other devices (and particularly those liable to be stolen) we will arrange for the duckling to die only when instructed by its mother: thus only the currently authorised user may transfer control of the device. In order to enforce this, some level of tamper resistance will be required: assassinating the duckling without damaging its body should be made suitably difficult and expensive. (Of course, there will be applications in which one wishes to protect against accidental death of the mother duck – such as if the remote control breaks. In such cases, we can make a 'backup mother' duck by squirrelling away a copy of the imprinting key.)

Some systems, such as Bluetooth, grant control to whoever has the ability to manipulate the device: if you can touch it, you can control it. The Duckling policy is different, because it specifies an element of tamper resistance to protect the crucial state transition of re-imprinting; that is, *resurrection control*. It follows that a passer-by cannot take ownership of an unattended duckling. So an imprinted car stereo is useless to a thief.

After a narrative illustration of the policy, we now list its four principles for reference.

1. (Two States) The entity that the policy protects, called the duckling, can be in one of two states: imprintable or imprinted (see figure 6). In the imprintable state, anyone can take it over. In the imprinted state, it only obeys another entity called its mother duck.
2. (Imprinting) The transition from imprintable to imprinted, known as imprinting, happens when the mother duck sends an imprinting

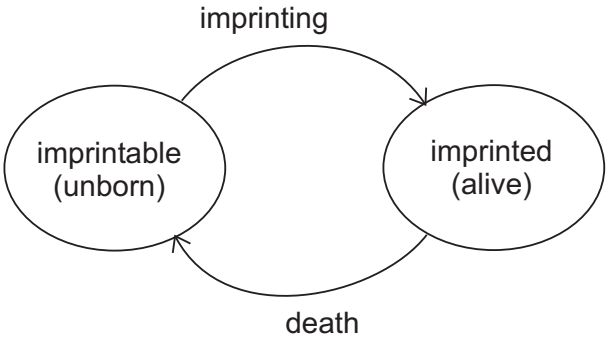

Figure 6: A state diagram for the Resurrecting Duckling .

key to the duckling. This may be done using physical contact, or by some other channel whose confidentiality and integrity are protected adequately.

3. (Death) The transition from imprinted to imprintable is known as death. It may only occur under a specified circumstance, such as death by order of the mother duck (default), death by old age (after a predefined time interval), or death on completion of a specific transaction.
4. (Assassination) The duckling must be uneconomical for an attacker to assassinate (which means to cause its death in circumstances other than as prescribed by the Death principle of the policy).

10 Access Control

As we have seen, security policies started primarily as coherent sets of constraints describing who could access what, and when. Even if our last few examples have shown more general scenarios, controlling access to resources tends to be the primary goal of many security policy models.

In the general case, given the set S of all the subjects in the system, and the set O of all the objects, we may build a classical two-dimensional access matrix with a row for each subject and a column for each object and where each cell of the matrix contains a Boolean value specifying whether that subject is allowed to access that object. In practice there will also be a third dimension to this matrix, indexed by the type of access (create, read, write, execute, browse, delete, etc.).

10.1 ACLs

Centralised administration of such a vast collection of individually significant security bits is difficult; to make the task more manageable, the matrix is usually split by columns or by rows.

When it is split by columns, to each object is associated a list of the subjects that are allowed to access it. This, appropriately enough, is called an Access Control List, or ACL (pronounced "ackle"). An example of this is given by the string of permission bits that Unix associates with each file. The sequence "rwxrwxrwx" represents three sets of "read, write, execute" permission bits, the sets respectively mapping to "user" (the owner of the

file), "group" (any subject in a designated group that has been associated with the file) and "other" (any other subject). If the permission is granted, the corresponding bit is set and is represented by its letter in the string; otherwise, the bit is reset and is represented by a hyphen instead of the letter.

It is important to note that these permission bits may be set by the owner of the file at her own discretion — hence the term "discretionary access control" to denote this situation as opposed to the "mandatory access control" of BLP and related multilevel secure systems in which no user can override the stated rules about information flow between clearance levels.

In general, mandatory access control makes it easier to guarantee that a stated security policy will be enforced. Central administration of a complex access matrix in constant flux as files and directories are created and destroyed is a very hard task, so mandatory access control usually implies a simplified view of the world, as in the BLP model, where users cannot specify all the individual access bits independently. Where finer control is necessary, discretionary access control is a manageable way to achieve greater flexibility. The two strategies may even be combined, as in Unix System V/MLS, where a base layer of mandatory access control is complemented by discretionary access control to further regulate accesses that are not constrained by the multilevel security rules.

10.2 Capabilities

If the access matrix is instead split by rows, we obtain for each subject a list of the objects to which she has access. The elements of such a list are called *capabilities*, meaning that if an object o is in the list for subject s, then s is capable of accessing o. In some systems, such as Windows 2000, there is also the concept of a "negative capability", to explicitly indicate that the given subject is not allowed to access the given object. There may be hierarchies of users, groups and directories through which a subject obtains both positive and negative capabilities for the same object and there will be rules that state which ones override the others.

To compare the interplay between the different approaches, observe the case of the BMA policy (section 7). Although it was naturally expressed in terms of access control lists, when it came to implementing it in a hospital the most natural expression was in capabilities or certificates. The majority of access control rules could be expressed in statements of the form 'a nurse may read, and append to, the records of all patients who have been in her ward during the previous 90 days'.

It should be noted that the concept of capability as a row of the access matrix is subtly different from the original one of a capability as "a bit string that you either know or don't know", as introduced by the Cambridge CAP machine [77] which implemented capabilities in hardware. There, each object was associated with an unguessable bit string (the capability) generated by the creator of the object; any subject wishing to access the object had to prove that it knew the capability[6]. Any subject with the right to access a given object could extend this right to another

[6]To clarify the difference between this idea of capability and the previous one, note that in this context a negative capability can't work. A negative capability would have to be a bit string that, if known, denies access to a resource (instead of granting it). Since it is up to the client to exhibit the capabilities he knows in order to be granted access, anyone given a negative capability might obviously find it convenient to just 'forget' it!

subject simply by telling it the capability. Even more interesting constructions were possible with the introduction of proxies (intermediate objects, acting as a layer of indirection, that among other things make revocation possible): instead of giving subject s the capability $c(o)$ of object o, the creator of o makes a proxy object p and gives $c(o)$ to p; it then gives s the capability $c(p)$ of the proxy, through which s can indirectly access o without knowing $c(o)$. The twist is that now the creator of o can revoke s's right to access o by simply deleting the proxy object — all without affecting the workings of any other subjects that might have been legitimately accessing o at the time. Capabilities fell into disuse in the 1980s and early 1990s, and were used only in a small number of systems, such as the IBM AS/400 series. They are now making a comeback in the form of public key certificates, which act as credentials for access to a set of resources. We'll discuss certificates below.

10.3 Roles

We also mentioned *role based access control*. Instead of assigning access rights (or, more generally, "privileges") to individual subjects such as Joe and Barbara, we assign them to roles such as "receptionist" and "personnel manager", and then give one or more roles to each subject. This is a very powerful organisational tool, in that it is much more meaningful to express a security target in terms of roles, which can be made to have well-defined and relatively stable semantics within the company, rather than in terms of individuals, whose functions (and employment status) may change over time.

To obtain the full benefits of such a scheme it is essential to maintain the distinction between subjects and roles, particularly when it comes to auditing. As an example of detrimental blurring of the two, consider the common case of the Unix "root" account on multiuser system with several administrators. Conceptually, "root" is a role, with which several subjects (the system administrators, who by the way have individual user accounts as well) are endowed. At the operating system level, however, "root" is only another subject, or user account, albeit a privileged one. This means that when, by malice or incompetence, one of the system administrators moves a dangerous file to an inappropriate place, the record of ownership and permissions will only say that this was done by "root", not by "Joe acting as root". Oracle in essence reimplements an entire user management and access control system on top of that provided by the underlying operating system, and seems to have got this right, with a clear separation between roles and users.

10.4 Security state

It must be noted that pure access control is not the best mechanism when the policy requires state to be retained. A subtle example of this comes from Clark-Wilson models that specify dual control, i.e., the fact that certain data items must be approved by two principals from different groups (say a yellow manager and a blue one) before becoming valid. Due to workflow constraints it is impractical to impose atomicity on the validation, since this would force the yellow and blue managers always to look at the transactions slips together. What is done instead is that the transaction slip is placed in the inbox of the yellow manager, gets approved by her at some point, then it moves to the inbox of the blue

manager, obtains its second approval and finally becomes valid. The system must therefore keep track of additional state for each Unconstrained Data Item to represent the approvals so far collected on the way to become a Constrained Data Item. Implementing a solution using only pure access control mechanisms (e.g. by creative use of intermediate files and directories with carefully chosen permissions and groups) is theoretically possible, but convoluted and error-prone. For example one has to prevent the blue manager from moving into the 'valid' directory a transaction that was never approved by yellow.

The problem is even more evident in the Chinese Wall case, where there is explicit mention of a Boolean matrix of state bits to indicate whether a consultant has ever interacted with any companies of a given type. Here too it is theoretically possible to pervert the file system's access control mechanisms into some means of artificially retaining the required state, such as by giving a consultant a positive capability for a company he decides to work for and simultaneously a negative capability for all the other companies in its conflict-of-interest class.

The cleaner alternative to these programming tricks is to keep the security state in a data structure explicitly devoted to that purpose. But where will such a data structure reside? Since it no longer implements a general purpose facility, like the permission bits of the file system, it is likely to migrate from the operating system into the application. The problem of *encapsulating* this data structure (in the object oriented sense) then arises: no other program must be able to modify the security state, except by using the methods provided by the application that is enforcing the policy. This is not always trivial to ensure.

11 Beyond Access Control

So not all security policies are elaborate sets of rules about access control. There are many contexts in which the aspect of greatest interest in the system is not access control but authentication, or delegation, or availability — or perhaps a combination of those and other properties. Biba and Jikzi are examples where integrity matters more than access control. These are not just a matter of controlling write access to files, as they bring in all sorts of other issues such as reliability, concurrency control and resistance to denial-of-service attacks.

On a more general level, we may speak of "security policy" whenever a consistent and unambiguous specification is drawn, stating the required behaviour of the system with respect to some specific security properties. Although we have presented a gallery of policy models and we have insisted on their strengths and general applicability, it is not necessary for a policy target to be derived as a specialisation of a model.

To clarify these points, let's examine a couple of examples of policies that are neither devoted to access control nor derived from established models.

11.1 Key management policies

Public key cryptography, as readers will know, is the technique introduced by Diffie and Hellman [30] whereby each principal is endowed with a pair of keys, one public and one private, whose associated cryptographic transformations are the inverse of each other. The public key is widely

disseminated while the private key is kept secret. This can be used for encryption or for digital signature. By publishing an encryption key and keeping the corresponding decryption key private, anyone can, using the public key, encrypt messages that only the holder of the private key will be able to decrypt. By publishing a signature verification key and keeping the corresponding signature creation key private, a principal can generate signatures that anybody can verify using the public key but which no other principal could have produced.

For such a system to work on a large scale, a way to manage and distribute public keys must be deployed. In particular, one must avoid the "man in the middle" attacks that become possible if malicious principals can convince their unsuspecting victims to accept forged public keys as those of their intended correspondents.

The CCITT X.509 recommendation [23], published in 1988, was the first serious attempt at such an infrastructure. It was part of the grander plan of X.500, a global distributed directory intended to assign a unique name to every principal (person, computer, peripheral, etc.) in the world — so called *Distinguished Names*. In this context, X.509 used *certificates* (i.e., signed statements) that bound unique names to public keys. Originally this was meant to control which principals had the right to modify which subtrees of X.500, but soon its use as an identity instrument became prevalent and it is used today to certify the public keys used with SSL, the protocol used for secure access to Web sites. Web sites wishing to accept credit card transactions typically have an encryption key certified by a company such as Verisign whose public key is well known; customers entering credit card numbers or other sensitive data can check the certificate to ensure that the public key with which the data will be encrypted is certified by Verisign to belong to the intended destination. X.509 is thus an example of a hierarchical public key infrastructure with a small number of global *roots* — master certification authorities on which all name certificates ultimately depend.

However the software that did most to bring public key cryptography into the mainstream was Zimmermann's *Pretty Good Privacy* — better known as simply PGP [80] — which has become the de facto standard for email encryption. One of PGP's conceptual innovations was the rejection of this hierarchical infrastructure of certification authorities in favour of a decentralised "web of trust" in which all the users, as peers, mutually certify the validity of the keys of their interlocutors. Users may thus obtain uncertified keys over insecure channels, as long as they can build "chains of trust" starting from people they know and leading to those keys.

There have been at least two attempts to get the best of both worlds. Ellison's Simple Public Key Infrastructure (SPKI), later to join forces with Rivest and Lampson's Simple Distributed Security Infrastructure (SDSI) [31, 32, 64], also rejected the concept of a single global certification authority. They bind keys directly to capabilities rather than via names. One of the core concepts is that of *local names* — identifiers that do not have to be globally unique as long as they are unique in the context in which they are used. Global names can be reintroduced as needed by placing a local name in the relevant context. So "Microsoft's public key" becomes "DNS's .com's Microsoft's public key", with DNS being a privileged context.

Without a single root, a user of the system must repeatedly make decisions on the validity of arbitrary keys and may at times be requested

to express a formal opinion on the validity of the key of another principal (by "signing" it). For consistency it is desirable that these actions be governed by a policy. Let us examine some examples — we shall refer to PGP for concreteness, since this is the most widely deployed system among end users.

The "signature of Alice on Bob's key" is actually a signature on the combination of Bob's public key and Bob's name. It means: "I, Alice, solemnly certify that, to the best of my knowledge, this key and this name do match". To Charlie, who must decide on the validity of Bob's key, this statement is only worth as much as Alice's reputation as an honest and competent introducer; in fact, PGP lets you assign a rating (denoted as "trust level") to each introducer, as well as a global confidence threshold that must be reached (by adding up the ratings of all the introducers) before a key can be considered as valid. For example you may request two signatures from marginally trusted introducers, or just one from a fully trusted one; but someone with a higher paranoia level might choose five and two respectively.

Such a rule would amount to a policy stating which keys to accept as valid. But the interesting aspects, as usual, come up in the details. A fundamental but easily neglected element of this policy would be a precise operational definition of when to classify an introducer as untrusted, marginally trusted or fully trusted.

The dual problem, equally interesting and probably just as neglected by individual users of PGP, is that of establishing a policy to govern one's signing of other people's keys. This is important if one wishes to be considered as a trustworthy introducer by others. One possible such policy might say:

1. I shall only certify a key if I have received or checked it in a face-to-face meeting with its owner.
2. I shall only certify a key if I have personally verified the passport of its owner.
3. Whenever I sign a key, I shall record date, key id and fingerprint in a signed log that I keep on my Web page.

Such a policy is known as a *certification practice statement*, and can offer some procedural guarantees about the quality of the certifications that one has performed. It gives an independent observer a chance to assess the relative quality of the certification offered by different introducers (assuming that their claims about compliance can be believed).

An observer could for example remark that the policy above, while apparently very strict, does not actually ascertain whether the named principal controls the private key corresponding to the public key being signed. Alice might follow the above policy and still sign a public key that Bob presents as his, but which he instead just copied off Charlie's Web page. This would not allow Bob to read Alice's (or anybody's) correspondence to Charlie, but it would enable him to damage Alice's reputation as a trustworthy introducer ("Look, she signed that this key belongs to Bob, but it's not true! She's too gullible to be an introducer!").

We might try to fix this hole by adding a challenge-response step to the policy: Alice shall only sign Bob's key if Bob is able to sign a random number chosen by Alice.

One lesson from all this is that policies, like ideas, tend to only become clear once we write them down in detail. It will be much harder to

spot a methodological flaw if the de facto policy has never been explicitly stated. This even applies to the above "fix": without a more explicit description of how to perform the challenge-response, it is impossible to say whether the proposed exchange is safe or still vulnerable to a "middleperson" attack. For example, Bob might offer to certify Charlie's key and simultaneously present it to Alice as his own for her to certify. She gives him a random challenge, he passes it to Charlie and Charlie provides the required signature. Bob now sends this signature to Alice who mistakenly certifies the key as Bob's.

Certification practice statements are even more important when we are dealing with a commercial or government certification authority rather than with private individuals using PGP. Such statements typically also set out the precise circumstances in which certificates will be revoked, and what liability (if any) will be borne for errors.

11.2 Corporate email

Another scenario calling for a policy unrelated to access control and not derived from one of the classical models is offered by corporate email. As the Microsoft trial demonstrated, a company can easily make the case against itself through the informal (but often revealing) messages that its executives exchange via email while discussing the campaign against their competitors.

A company aware of this precedent, and concerned that its off-the-record internal messages could be used against it, might decide to get rid of them at the first sign of trouble. But this could be very risky for a company already under investigation, as a court could punish it for contempt or obstruction.

So a company may establish a policy to delete all email messages older than, say, one year. If the company has information in its archives, a court might force its disclosure; if the company deletes the information once the investigation has started, it is at risk; but if the company has an explicit established policy of not archiving old email, then it cannot be faulted for not being able to produce old correspondence.

Another example comes from our university. Examiners are required to destroy exam papers, working notes and other documents after four months. If they were kept too long, students could acquire the right to see them under data protection law, and this would violate our confidentiality policy; but destroying them too soon might prejudice appeals.

A policy of timely destruction has to address a number of practical issues (such as system backups and data on local discs), which makes its implementation nontrivial. A simple technique might involve encrypting all messages before storing them using a company-wide limited-lifetime key held in a tamper resistant box that deletes it after the specified time interval. Efficient purging of unreadable messages is left as a garbage collection task for the system administrator.

Of course none of this stops people from taking copies of messages while they are still readable, but:

1. If we wanted to stop that, we would need further and more complicated mechanisms to implement the policy.
2. It would never be possible to implement such a policy in a completely watertight manner: after all, a determined employee could always print out the mail or, if even that were forbidden, photograph the screen.

3. We probably don't want that anyway: it would be wrong to see the legitimate user reading email as the potential enemy when the real one is his carelessness. Taking a permanent copy of an important message should be allowed, as long as this exceptional action requires explicit confirmation and is audited.

At any rate, apart from the implementation details, the point we want to emphasise here is the use of a policy as a legal defence. The security property of interest in this case is not access control but **plausible deniability**. (It's not really a matter of "no-one may read X after time T" but "even the system administrator will not be able to create a user who can".)

12 Automated Compliance Verification

Once policy is refined from a general model to a specific target, there is interest in a system that would automatically verify whether any given proposed action is acceptable.

Blaze, Feigenbaum and Lacy introduced the concept of trust management [19]: a unified approach to specifying security policies and credentials and to verifying compliance. The system they proposed and built, PolicyMaker, includes an application-independent engine whose inputs are policy assertions, security credentials (i.e., "certificates") and the proposed action, and whose output is a series of "acceptance records" that say which assertions, if any, authorise the action. The idea is for this generic engine to be configured by an appropriate application-specific policy. Requests for security-critical actions must be accompanied by appropriate credentials in order to satisfy the system that the principal issuing the request has the authority to do so.

Related work includes SPKI and SDSI, which address not only authorisation but also naming, i.e. the association of identities to public keys. PolicyMaker explicitly refuses to deal with the problem of naming; its authors argue that it is orthogonal to authorisation and therefore irrelevant to the issue of compliance checking.

The successor to PolicyMaker, called KeyNote, is now RFC 2704 [18]. PolicyMaker was designed for generality as a framework in which to explore trust management concepts, perhaps at the expense of efficiency. For example the assertions could be arbitrary programs. KeyNote is rather designed for simplicity, competitiveness and efficiency, with the aim of being fielded in real applications. Popular open source projects including the Apache-SSL Web server and the OpenBSD operating system already use KeyNote.

13 A Methodological Note

As a final exhibit in this gallery of examples it is worth mentioning our study of the security requirements for a computer-based National Lottery system [5]. More than the security policy model in itself, what is most instructive in this case is methodology employed for deriving it.

Experienced software engineers know that perhaps 30% of the cost of a software product goes into specifying it, 10% into coding, and the remaining 60% on maintenance.

Specification is not only the second most expensive item in the system development life cycle, but is also where the most expensive things go wrong. The seminal study by Curtis, Krasner and Iscoe of large software project disasters found that failure to understand the requirements was mostly to blame [25]: a thin spread of application domain knowledge typically led to fluctuating and conflicting requirements, which in turn caused a breakdown in communication. They suggested that the solution was to find an 'exceptional designer' with a deep understanding of the problem who would assume overall responsibility.

But there are many cases where an established expert is not available, such as when designing a new application from scratch or when building a competitor to a closed, proprietary system whose behaviour can only be observed at a distance. It therefore seemed worthwhile to see if a high quality security specification could be designed in a highly parallel way, by getting a lot of different people to contribute drafts in the hope that most of the possible attacks would be considered in at least one of them.

We carried out such an experiment in 1999 by recruiting volunteers from a captive audience of final year undergraduates in computer science at the University of Cambridge: we set one of their exam questions to be the definition of a suitable security policy for a company planning to bid for the licence to the British National Lottery.

The model answer had a primary threat model that attackers, possibly in cahoots with insiders, would try to place bets once the result of the draw was known, whether by altering bet records or forging tickets. The secondary threats were that bets would be placed that had not been paid for, and that attackers might operate bogus vending stations that would pay small claims but disappear if a client won a big prize.

The security policy that follows logically from this is that bets should be registered online with a server that is secured prior to the draw, both against tampering and against the extraction of sufficient information to forge a winning ticket; that there should be credit limits for genuine vendors; and that there should be ways of identifying bogus vendors. Once the security policy has been developed in enough detail, designing enforcement mechanisms should not be too hard for someone skilled in the computer security art.

Valuable and original contributions from the students came at a number of levels, including policy goal statements, discussions of particular attacks, and arguments about the merits of particular protection mechanisms.

At the level of goals, for example, one candidate assumed that the customer's rights must have precedence: "All winning tickets must be redeemable! So failures must not allow unregistered tickets to be printed." Another candidate assumed the contrary, and thus the "worst outcome should be that the jackpot gets paid to the wrong person, never twice." Such goal conflicts are harder to identify when the policy goals are written by a single person.

As for attacks, some candidates suggested using the broadcast radio clock signal as an authentication input to the vending terminals; but one candidate correctly pointed out that this signal could be jammed without much difficulty. This caused some consternation to the auditor of a different online gaming system, which appears to be vulnerable to time signal spoofing.

There was a lot of discussion not just on how to prevent fraud but how to assure the public that the results were trustworthy, by using tech-

niques such as third party logging or digital signatures. The candidates' observations on protection mechanisms also amounted to a very complete checklist. Items such as 'tickets must be associated with a particular draw' might seem obvious, but a protocol design that used a purchase date, ticket serial number and server-supplied random challenge as input to a MAC computation might appear plausible to a superficial inspection. The evaluator might not check to see whether a shopkeeper could manufacture tickets that could be used in more than one draw. Experienced designers appreciate the value of such checklists.

The lesson drawn from this case study was that requirements engineering, like software testing and unlike software development, is susceptible to parallelisation. When developing the threat analysis, security requirements and policy model for a new system, rather than paying a single consultant to think about a problem for twenty days, it will often be more efficient to pay fifteen consultants to think about it for a day each and then have an editor spend a week hammering their ideas into a single coherent document.

14 Conclusions

A security policy is a specification of the protection goals of a system. Many expensive failures are due to a failure to understand what the system security policy should have been. Technological protection mechanisms such as cryptography and smartcards may be more glamorous for the implementer, but technology-driven designs have a nasty habit of protecting the wrong things.

At the highest level of abstraction, a security policy model has little if any reference to the mechanisms that will be used to implement it. At the next level down, a protection profile sets out what a given type of system or component should protect, without going into implementation detail, and relates the protection mechanisms to threats and evironmental assumptions. A security target gives a precise statement of what a given system or component will protect and how. Especially at the highest levels the policy functions as a means of communication. Like any specification, it is a contract between the implementer and the client — something that both understand and by which both agree to be bound.

Our historical perspective has shown how security policies were first formally modelled in the 1970s to manage disclosure threats in military systems. They were then extended to issues other than confidentiality and to problems other than access control. We have also seen a spectrum of different formulations, from the more mathematically oriented models that allow one to prove theorems to informal models expressed in natural language. All have their place. Often the less formal policies will acquire more structure once they have been developed into protection profiles or security targets and the second- and third-order consequences of the original protection goals have been discovered.

We now have a sufficiently large gallery of examples, worked out in varying levels of detail, that when faced with a project to design a new system, the security engineer should first of all assess whether she can avoid reinventing the wheel by adopting one of them. If this is not possible, familiarity with previous solutions is always helpful in coming up with an appropriate new idea. Finally, the methodological issues should not be underestimated: security always benefits from peer review and many

heads are better than one.

15 Acknowledgements

The authors are grateful to Jeremy Epstein, Virgil Gligor, Paul Karger, Ira Moskowitz, Marv Schaefer, Rick Smith, Karen Spärck Jones and Simon Wiseman for helpful discussions.

Portions of this chapter will appear in Ross Anderson's book *Security Engineering* [8], to which the reader should refer for more detail. Other portions have appeared in Jong-Hyeon Lee's PhD dissertation [49] and in other publications by the authors that were cited in the relevant sections [10, 11, 12, 68, 69, 5].

References

[1] Edward Amoroso. *Fundamentals of Computer Security Technology*. Prentice-Hall, Englewood Cliffs, New Jersey, 1994. ISBN 0-13-305541-8.

[2] J. Anderson. "Computer Security Technology Planning Study". Tech. Rep. ESD-TR-73-51, AFSC, Hanscom AFB, Bedford, MA, Oct 1972. AD-758 206, ESD/AFSC.

[3] M. Anderson, C. North, J. Griffin, R. Milner, J. Yesberg and K. Yiu. "Starlight: Interactive Link". In "12th Annual Computer Security Applications Conference", pp. 55–63. IEEE, 1996. ISBN 0-8186-7606-X.

[4] Ross Anderson. "A Security Policy Model for Clinical Information Systems". In "Proceedings of the IEEE Symposium on Research in Security and Privacy", Research in Security and Privacy, pp. 30–43. IEEE Computer Society,Technical Committee on Security and Privacy, IEEE Computer Society Press, Oakland, CA, May 1996.

[5] Ross Anderson. "How to Cheat at the Lottery (or, Massively Parallel Requirements Engineering)". In "Proceedings of the Annual Computer Security Applications Conference 1999", Phoenix, AZ, USA, 1999. URL `http://www.cl.cam.ac.uk/~rja14/lottery/lottery.html`.

[6] Ross J. Anderson. "The DeCODE Proposal for an Icelandic Health Database". *Læknabladh idh* (The Icelandic Medical Journal), **84**(11):874–875, Nov 1998. URL `http://www.cl.cam.ac.uk/users/rja14/iceland/iceland.html`. The printed article is an excerpt from a document produced for the Icelandic Medical Association. The full text of the latter is available online.

[7] Ross J. Anderson. "Comments on the Security Targets for the Icelandic Health Database", 1999. URL `http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/iceland-admiral.p%df`.

[8] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2001. ISBN 0-471-38922-6.

[9] Ross John Anderson. "Why Cryptosystems Fail". *Communications of the ACM*, **37**(11):32–40, 1994.

[10] Ross John Anderson. "Security in Clinical Information Systems". BMA Report, British Medical Association, Jan 1996. ISBN 0-7279-1048-5.

[11] Ross John Anderson and Jong-Hyeon Lee. "Jikzi: A New Framework for Secure Publishing". In "Proceedings of Security Protocols Workshop '99", Cambridge, Apr 1999.

[12] Ross John Anderson and Jong-Hyeon Lee. "Jikzi – A New Framework for Security Policy, Trusted Publishing and Electronic Commerce". *Computer Communications*, to appear.

[13] L. Badger, D. F. Sterne, D. L. Sherman, K. M. Walker and S. A. Haghighat. "Practical Domain and Type Enforcement for UNIX". In "Proceedings of the 5th USENIX UNIX Security Symposium", pp. 66–77. Oakland, CA, May 1995.

[14] D. Elliot Bell and Leonard J. LaPadula. "Secure Computer Systems: Mathematical Foundations". Mitre Report ESD-TR-73-278 (Vol. I–III), Mitre Corporation, Bedford, MA, Apr 1974.

[15] T Benkart and D Bitzer. "BFE Applicability to LAN Environments". In "Seventeenth National Computer Security Conference", pp. 227–236. NIST, Baltimore, Maryland, 11–14 Oct 1994.

[16] T. Berson and G. Barksdale. "KSOS-Development Methodology for a Secure Operating System". In "Proc. NCC", pp. 365–371. AFIPS, AFIPS Press, Montvale, NJ, Jun 1979. Vol. 48.

[17] Ken Biba. "Integrity Considerations for Secure Computing Systems". Mitre Report MTR-3153, Mitre Corporation, Bedford, MA, 1975.

[18] Matt Blaze, Joan Feigenbaum, John Ioannidis and A. Keromytis. "The KeyNote Trust-Management System Version 2". IETF RFC 2704, Internet Engineering Task Force, Sep 1999. URL `http://www.cis.ohio-state.edu/htbin/rfc/rfc2704.html`.

[19] Matt Blaze, Joan Feigenbaum and Jack Lacy. "Decentralized Trust Management". In "Proceedings of the IEEE Symposium on Research in Security and Privacy", Research in Security and Privacy. IEEE Computer Society,Technical Committee on Security and Privacy, IEEE Computer Society Press, Oakland, CA, May 1996.

[20] W. E. Boebert and R. Y. Kain. "A Practical Alternative to Hierarchical Integrity Policies". In "Proceedings of the 8th National Computer Security Conference", pp. 18–27. NIST, 1985.

[21] David F. C. Brewer and Michael J. Nash. "The Chinese Wall Security Policy". In "1989 IEEE Symposium on Security and Privacy", pp. 206–214. Oakland, CA, 1989.

[22] C. Cant and S. Wiseman. "Simple Assured Bastion Hosts". In "13th Annual Computer Security Application Conference", pp. 24–33. IEEE Computer Society, 1997. ISBN 0-8186-8274-4.

[23] CCITT. "Data Communications Networks Directory". Tech. Rep. 8, CCITT, Melbourne, Nov 1988. Recommendations X.500-X.521, IXth Plenary Assembly.

[24] David D. Clark and David R. Wilson. "A Comparison of Commercial and Military Computer Security Policies". In "1987 IEEE Symposium on Security and Privacy", pp. 184–194. Oakland, CA, 1987.

[25] Bill Curtis, Herb Krasner and Neil Iscoe. "A Field Study of the Software Design Process for Large Systems". *Communications of the ACM*, **31**(11):1268–1287, Nov 1988.

[26] I. Denley and S. Weston-Smith. "Privacy in clinical information systems in secondary care". *British Medical Journal*, **318**:1328–1331, May 1999.

[27] Dorothy E. Denning. "A Lattice Model of Secure Information Flow". *Communications of the ACM*, **19**(5):236–243, May 1976. ISSN 0001-0782. Papers from the Fifth ACM Symposium on Operating Systems Principles (Univ. Texas, Austin, Tex., 1975).

[28] Dorothy E. Denning. "A Lattice Model of Secure Information Flow". *Communications of the ACM*, **19**(5):236–243, May 1976. ISSN 0001-0782.

[29] Dorothy E. R. Denning. *Cryptography and Data Security*. Addison-Wesley, Reading, 1982. ISBN 0-201-10150-5.

[30] Whitfield Diffie and Martin E. Hellman. "New directions in cryptography". *IEEE Transactions on Information Theory*, **IT-22**(6):644–654, 1976.

[31] Carl Ellison. "The nature of a useable PKI". *Computer Networks*, **31**(8):823–830, May 1999.

[32] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas and Tatu Ylonen. "SPKI Certificate Theory". IETF RFC 2693, Internet Engineering Task Force, Sep 1999. URL `http://www.cis.ohio-state.edu/htbin/rfc/rfc2693.html`.

[33] J Epstein, H Orman, J McHugh, R Pascale, M Branstad and A Marmor-Squires. "A High Assurance Window System Prototype". *Journal of Computer Security*, **2**(2–3):159–190, 1993.

[34] J Epstein and R Pascale. "User Interface for a High Assurance Windowing System". In "Ninth Annual Computer Security Applications Conference", pp. 256–264. IEEE, Orlando, Florida, USA, 6–10 Dec 1993. ISBN 0-8186-4330-7.

[35] Glenn Faden. "Reconciling CMW Requirements with Those of X11 Applications". In "Proceedings of the 14th Annual National Computer Security Conference", Washington, DC, USA, Oct 1991. Architecture of the windowing portion of Sun's CMW.

[36] JS Fenton. *Information Protection Systems*. Phd dissertation, Cambridge University, 1973.

[37] D. Ferraiolo and R. Kuhn. "Role-Based Access Controls". In "15th NIST-NCSC National Computer Security Conference", pp. 554–563. Oct 1992.

[38] Simon N. Foley. "Aggregation and separation as noninterference properties". *Journal of Computer Security*, **1**(2):158–188, 1992.

[39] L. J. Fraim. "SCOMP: A Solution to the Multilevel Security Problem". *Computer*, **16**(7):26–34, Jul 1983.

[40] T. Fraser. "LOMAC: Low Water-Mark Integrity Protection for COTS Environments". In "Proceedings of the 2000 IEEE Symposium on Security and Privacy", pp. 230–245. IEEE Computer Society Press, 2000.

[41] J. A. Goguen and J. Meseguer. "Security Policies and Security Models". In "Proceedings of the 1982 Symposium on Security and Privacy (SSP '82)", pp. 11–20. IEEE Computer Society Press, Los Alamitos, Ca., USA, Apr 1990.

[42] R. D. Graubart, J. L. Berger and J. P. L. Woodward. "Compartmented Mode, Workstation Evaluation Criteria, Version 1". Tech. Rep. MTR 10953 (also published by the Defense Intelligence Agency as document DDS-2600-6243-91), The MITRE Corporation, Bedford, MA, USA, Jun 1991. Revised requirements for the CMW, including a description of what they expect for Trusted X.

[43] Michael A. Harrison, Walter L. Ruzzo and Jeffrey D. Ullman. "Protection in Operating Systems". *Communications of the ACM*, **19**(8):461–471, Aug 1976. ISSN 0001-0782.

[44] G Huber. "CMW Introduction". *ACM SIGSAC*, **12**(4):6–10, Oct 1994.

[45] M. H. Kang and I. S. Moskowitz. "A Pump for Rapid, Reliable, Secure Communications". In ACM (ed.), "Fairfax 93: 1st ACM Conference on Computer and Communications Security, 3–5 November 1993, Fairfax, Virginia", pp. 118–129. ACM Press, New York, NY 10036, USA, 1993. ISBN 0-89791-629-8.

[46] MH Kang, JN Froscher and IS Moskowitz. "An Architecture for Multilevel Secure Interoperability". In "13th Annual Computer Security Applications Conference", pp. 194–204. IEEE Computer Society, San Diego, CA, USA, 8–12 Dec 1997. ISBN 0-8186-8274-4.

[47] MH Kang, IS Moskowitz, B Montrose and J Parsonese. "A Case Study of Two NRL Pump Prototypes". In "12th Annual Computer Security Applications Conference", pp. 32–43. IEEE, San Diego CA, USA, 9–13 Dec 1996. ISBN 0-8186-7606-X.

[48] P.A. Karger, V.A. Austell and D.C. Toll. "A New Mandatory Security Policy Combining Secrecy and Integrity". Tech. Rep. RC 21717 (97406), IBM, Mar 2000.

[49] Jong-Hyeon Lee. "Designing a reliable publishing framework". Tech. Rep. 489, University of Cambridge Computer Laboratory, Apr 2000.

[50] Mark Lomas. "Auditing against Multiple Policies (Transcript of Discussion)". In "Proceedings of Security Protocols Workshop 1999", No. 1796 in Lecture Notes in Computer Science, pp. 15–20. Springer-Verlag, Apr 1999.

[51] Konrad Lorenz. *Er redete mit dem Vieh, den Vögeln und den Fischen* (King Solomon's ring). Borotha-Schoeler, Wien, 1949.

[52] Daryl McCullough. "A Hookup Theorem for Multilevel Security". *IEEE Transactions on Software Engineering*, **16**(6):563–568, Jun 1990. ISSN 0098-5589. Special Section on Security and Privacy.

[53] J McLean. "Security Models". In "Encyclopedia of Software Engineering", John Wiley & Sons, 1994.

[54] John McLean. "A comment on the 'basic security theorem' of Bell and LaPadula". *Information Processing Letters*, **20**(2):67–70, Feb 1985. ISSN 0020-0190.

[55] D.P. Moynihan. *Secrecy — The American Experience*. Yale University Press, 1999. ISBN 0-300-08079-4.

[56] Paul Mukherjee and Victoria Stavridou. "The Formal Specification of Safety Requirements for Storing Explosives". *Formal Aspects of Computing*, **5**(4):299–336, 1993.

[57] M Nash and R Kennett. "Implementing Security policy in a Large Defence Procurement". In "12th Annual Computer Security Applications Conference", pp. 15–23. IEEE, San Diego, CA, USA, 9–13 Dec 1996. ISBN 0-8186-7606-X.

[58] National Security Agency. "The NSA Security Manual". Tech. rep., NSA. URL `http://www.cl.cam.ac.uk/ftp/users/rja14/nsaman.tex.gz`. (Leaked copy.).

[59] Roger Michael Needham and Michael Schroeder. "Using Encryption for Authentication in Large Networks of Computers". *Communications of the ACM*, **21**(12):993–999, 1978.

[60] B. Clifford Neuman and John T. Kohl. "The Kerberos Network Authentication Service (V5)". IETF RFC 1510, Internet Engineering Task Force, Sep 1993.

[61] NIST. "Common Criteria for Information Technology Security, Version 2.1". Tech. Rep. ISO IS 15408, National Institute of Standards and Technology, Jan 2000. URL `http://csrc.nist.gov/cc/`.

[62] Public Record Office. "Functional Requirements for Electronic Record Management Systems", Nov 1999. URL `http://www.pro.gov.uk/recordsmanagement/eros/invest/reference%.pdf`.

[63] B Pomeroy and S Wiseman. "Private Desktops and Shared Store". In "Computer Security Applications Conference", pp. 190–200. IEEE, Phoenix, AZ, USA, 1998. ISBN 0-8186-8789-4.

[64] Ronald L. Rivest and Butler W. Lampson. *SDSI – A Simple Distributed Security Infrastructure*, Apr 1996. URL `http://theory.lcs.mit.edu/~cis/sdsi.html`. V1.0 presented at USENIX 96 and Crypto 96.

[65] J. Rushby and B. Randell. "A Distributed Secure System". In "IEEE Computer", pp. 55–67. IEEE, Jul 1983.

[66] RR Schell. "Computer Security: The Achilles' Heel of the Electronic Air Force?" *Air University Review*, **30**(2):16–33, Jan–Feb 1979.

[67] RR Schell, PJ Downey and GJ Popek. "Preliminary notes on the design of secure military computer systems". Tech. Rep. MCI-73-1, Electronic Systems Division, Air Force Systems Command, 1 Jan 1973. URL `http://seclab.cs.ucdavis.edu/projects/history/papers/sche73.p%df`.

[68] Frank Stajano and Ross Anderson. "The Resurrecting Duckling: Security Issues in Ad-Hoc Wireless Networks". In Bruce Christianson, Bruno Crispo and Mike Roe (eds.), "Security Protocols, 7th International Workshop Proceedings", Lecture Notes in Computer Science. Springer-Verlag, 1999. URL `http://www.cl.cam.ac.uk/~fms27/duckling/`. Also available as AT&T Laboratories Cambridge Technical Report 1999.2.

[69] Frank Stajano and Ross Anderson. "The Resurrecting Duckling: Security Issues in Ad-Hoc Wireless Networks". In "Proceedings of 3^{rd} AT&T Software Symposium", Middletown, New Jersey, USA, Oct 1999. URL `http://www.cl.cam.ac.uk/~fms27/duckling/`. Abridged and revised version of the Security Protocols article by the same name. Also available as AT&T Laboratories Cambridge Technical Report 1999.2b.

[70] David Sutherland. "A Model of Information". In "Proc. 9th National Security Conference", pp. 175–183. Gaithersburg, Md., 1986.

[71] US Department of Defense. "Technical Rationale behind CSC-STD-003-85: computer security requirements". Tech. Rep. CSC-STD-004-85, US Department of Defense, 1985.

[72] US Department of Defense. "Trusted Computer System Evaluation Criteria". Tech. Rep. 5200.28, US Department of Defense, 1985.

[73] KG Walter, WF Ogden, WC Rounds, FT Bradshaw, SR Ames and DG Shumway. "Models for Secure Computer Systems". Tech. Rep. 1137, Case Western Reserve University, 31Jul 1973. Revised 21 Nov 1973.

[74] KG Walter, WF Ogden, WC Rounds, FT Bradshaw, SR Ames and DG Shumway. "Primitive Models for Computer Security". Tech. Rep. ESD–TR–74–117, Case Western Reserve University, 23Jan 1974. URL `http://www.dtic.mil`.

[75] Clark Weissman. "Security Controls in the ADEPT-50 Time-Sharing System". In "Proc. Fall Joint Computer Conference, AFIPS", vol. 35, pp. 119–133. 1969.

[76] Clark Weissman. "BLACKER: Security for the DDN, Examples of A1 Security Engineering Trades". In "Proceedings of the 1992 IEEE Computer Society Symposium on Security and Privacy (SSP '92)", pp. 286–292. IEEE, May 1992. ISBN 0-8186-2825-1.

[77] M. V. Wilkes and R. M. Needham (eds.). *The Cambridge Cap Computer and its Operating System*. North-Holland, New York, 1979. ISBN 0-444-00357-6.

[78] J. P. L. Woodward. "Security Requirements for System High and Compartmented Mode Workstations". Tech. Rep. MTR 9992, Revision 1 (also published by the Defense Intelligence Agency as document DDS-2600-5502-87), The MITRE Corporation, Bedford, MA, USA, Nov 1987. The original requirements for the CMW, including a description of what they expect for Trusted X.

[79] P Wright. *Spycatcher – The Candid Autobiography of a Senior Intelligence Officer*. William Heinemann Australia, 1987. ISBN 0-85561-098-0.

[80] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, 1995. ISBN 0-262-74017-6.