The Pesto Project
–Goals and Motivation–

Feike W. Dillema and Tage Stabell-Kulø

Technical Report 2001-40

June 2001

1 Introduction

We can say that a *personal* computer is one that is available to the user when he wants to use it, while a *private* computer is one that is never available to others. We use the term *private computing* to describe the computational model which involves small, private, hand-held computers, i.e. Personal Digital Assistants (PDAs). In general, a PDA offers its user relatively few resources; a low-bandwidth user-interface, (relatively) low computational power, and highly variable communication resources. However, the PDA excels in two aspect: *Trust* and *Availability*.

A PDA that is carried by its owner has great potential to be his private computer, an attractive sanctuary for private data and computations (such as creating digital signatures). Availability means that the PDA can be used in new settings, but availability also means physical control over the device. Physical control is a prerequisite for secure operation of almost any device. With trust in the private computer itself in place, the challenge of private computing is to use a PDA as solid ground from where to extend a user's sphere of control to distributed and less-trusted resources he cares to use in a variety of different settings.

The trend towards mobile computing adds the requirement that users should be able to roam between many different environments. Independent of what administrative domain they roam into, they want access to their personal resources while using resources available in that domain. As a general rule, we can say that policies, mechanisms and resources are different in each environment and system, and a lot of machinery between systems is necessary in order to maintain and achieve important nonfunctional aspects like security and safety across systems. However, creating this machinery is often impossible because policies and available mechanisms in the different domains are too diverse. We believe that ubiquitous computing requires a platform for distributed systems and applications that makes no assumptions on the environment the user currently is in.

There are basically two possible ways to support private computing. One can either build all the required machinery into the applications, or one can provide the applications with an underlying infrastructure. The latter approach leaves it to the applications to *adapt to* the system state as presented to them, rather than letting applications *detect* this distributed state themselves. Whether an infrastructure is viable and efficient depends on whether it supports the least common dominator of the applications' needs.

We believe that most applications operating in our target environment share the need for highly available, secure storage of data. We argued in [7] that systems for private computing should be designed in accordance with the *open-end argument*, which states that many of the nonfunctional aspects in a distributed system can only be reasonably quantified by the user (for example, who and what to trust). Qualitative assessment of information should be done by users only, and the system should be designed as to make this feasible for them to do so.

These observations formed the motivation for the design of the Pesto storage infrastructure. Pesto incorporates basic, but flexible, safety and security mechanisms. It is designed to be user-centric and to regard the user as the ultimate source of qualitative assessments and policy decisions.

2 Design Goals

The challenge of private computing is to design a system that allows its users to span other systems in such way that they retain ownership and control over their own information, without requiring administrative control over these systems. To achieve this goal, Pesto separates trust relations from administrative relations:

- Pesto has facilities that enable a user to control his data and enforce his real-world ownership. The user is the only one that can specify security and safety policies *and* the only one that decides who is trusted to enforce these. Pesto allows a user to delegate authority to other nodes trusted by him, but he need not do so. If the user does not authorize some part of the system to speak on his behalf on a certain matter, Pesto will involve the user himself in the decision process to resolve questions concerning it.

- Pesto supports recovery by storing and replicating all authorized updates and recording who authorized them. Experience shows that users are often unwilling to pay any non-trivial price for prevention of a risk, for as long as they have not been negatively affected by it [5]. This is well documented in the security and safety engineering literature as a very common cause for security breeches and calamities (see e.g. [1, 6]). In addition, real-world trust relationship are volatile and may change suddenly. We assume that disasters *will* happen; nodes will fail, trust will be violated, subtle mistakes will be made and accidents will occur.

- Pesto makes it possible for user Bob to share his content with another user Alice, regardless of whether Alice is known to Pesto. She does not need to establish an administrative relationship with his administrative domain beforehand. He can delegate authority to her in the manner he feels appropriate, e.g. by using delegation certificates based on public-key cryptography. However, public-key cryptography is not a fundamental building block in the system such that delegation of authority in Pesto does not depend on it. Furthermore, Bob not only defines his access control policies but also decides who is trusted to enforce these.

- Pesto provides the means for users to share data in different settings. It is well known that sharing of data in typical file systems is relatively rare, see e.g. [4, 3]. The more private data is, sharing between different users is less likely, and one could assume that in private computing sharing would be close to non-existing. However, Pesto assumes that *the same* user, e.g. at different locations, using different machines or in different rôles, will cause private data to be shared. Also, when data *is* explicitly shared between two users, conflicts will indeed be common. All in all, conflicts will be more common than in traditional systems.

  Sharing of data in a distributed system requires some form of consistency control to resolve conflicting concurrent updates. Consistency can be enforced at the system-level or the application-level. A system-level implementation would require the user to trust a subset of the system to run a consistency control protocol on his behalf. Pesto avoids such requirements by separating replication control from consistency control.

- Pesto separates storage and availability of *data* (encrypted content) from accessibility of *content*. Assume that Alice owns some storage space. She controls this resource and decides who may make use of it. But when she grants Bob the right to use some of this storage, the *system* should not require of him that he makes the *content* available to her as well.

  Sometimes a user will have no other option however, than to trust another user with some content in order to make use of his resources, even though there might not be a basis for such trust. For example, a user roaming into unknown territory carrying merely his PDA may find himself lacking enough trusted resources to accomplish his task and his need to make progress may be greater than his need for privacy for that task. Editing a file on an untrusted machine should not require trust in other infrastructure, like the network links between that machine and the user's machine at home. In addition, it should not put the privacy of any other content of the user at risk. Pesto allows a user to put his privacy at risk in order to make work progress, while minimizing the risk involved.

- Pesto supports both disconnected and semi-disconnected operation by using asynchronous communication and computation throughout its design. Disconnected operation remains a common mode of operation for private machines, regardless of the ever growing network coverage. Also, low bandwidth and/or relatively expensive communication (like GSM data) prevails, especially for mobile machines.

3 Separation of Concerns

The design of Pesto carefully separates the different mechanisms a distributed storage infrastructure must support. Because of this, a user can place responsibility of the different tasks on different machines, possibly governed by a variety of administrative domains. In this way, the user is free to set up relationships he is comfortable with and assign tasks to different parts of the infrastructure. As the primary tasks in our system are providing storage, replication and access to resources, we define the responsible parts of the infrastructure accordingly:

Trusted Storage Base (TSB): The set of servers a user trusts to store replicas of data (encrypted content). These servers are the storage service providers of the user. A user may define a different TSB for different sets of data.

Trusted Replicator Base (TRB): The set of servers that have been given authority over a part of the user's storage resources. The servers in a TRB are trusted to enforce a replication policy on behalf of the user. Each member of a TRB is responsible for the distribution of replicas to some subset of servers in a TSB. Together they make up a directed distribution graph with edges leaving only servers in the TRB and ending in servers in the TSB. A user may define a different TRB for different sets of data.

Trusted Access Base (TAB): The set of servers that have been given authority over a part of the user's content. The members of a TAB are trusted to enforce an access control policy on behalf of the user. A user may define a different TAB for different sets of data. Typically, TAB members are also members of the TRB.

In addition to this separation in space, Pesto also separates tasks in time in order to support (full and semi) disconnected operation:

Replication: Distribution of replicas is performed using an asynchronous request-response protocol such that request and response messages may be separated by a period of disconnected operation.

Access Control: Authorization for access to content and storage can be acquired at a different time than their actual usage, allowing for work progress when disconnected from the relevant trusted access base.

The infrastructure is assumed to offer best-effort service only. Monitoring and control of the quality of the actual (storage and replication) service is left to the user and his (management) applications. The infrastructure is not responsible for consistency control. That task is separated from replication control and is left to the applications and the user himself. The basic Pesto protocol is completely asynchronous, but supports so-called guarding of messages as synchronization primitive for applications to use as building block for their own consistency control mechanisms. A guarded message lists a number of other messages as its guards. A message will not be processed by its recipient untill its guards have been processed. Pesto's guarding of messages allows for the construction of consistency protocols (aiming at e.g. quorum-consensus for updates) on top of the basic Pesto protocol.

During a network partition it is impossible to know from within the system in the minority partition, whether it is safe to proceed with a task that alters shared data [2]. Strong consistency requirements are therefore impossible or very expensive to meet in Pesto's target environment, where the network may never be free of partitions. Supporting disconnected operation is essential such that optimistic replication is the only feasible option for Pesto. Consequence of this is that merging of conflicting updates may be required upon reconnection.

Whether independent updates represent a conflict is application dependent; e.g. an append-only unordered log will see no conflicts. For application content, conflict resolution semantics can not be defined by the storage system without limiting the range of application requirements that it can support efficiently. The user, on the other hand, may be able to *evaluate* the risk of disconnected operation and understand the possible consequences of his actions. It should be *his* task to decide whether being able to make work progress now is worth the risk of conflict resolution work later. It is not the task of the storage system to make such an assessment, and Pesto therefore will not deny a user service based on some built-in notion of how to preserve correctness. Pesto allows an application to create an update, while delaying (possibly forever) storing its content, which can be used to implement advisory locking schemes that work well even in a semi-partitioned network.

References

[1] R. Anderson. Why cryptosystems fail. *Communications of the ACM*, 37(11):32–41, November 1994.

[2] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of 7th SOSP*, pages 150–62. ACM Press, 1979.

[3] Dorota M. Huizinga and Ken A. Heflinger. Experience with connected and disconnected operation of portable notebook computers in distributed systems. In *Proceedings of the 1st IEEE Workshop on Mobile Computing Systgems and Applications*, pages 119–23. IEEE, December 1994.

[4] James J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[5] Nancy G. Leveson. *Safeware; System Safety and Computers*. Addison Wesley, 1995.

[6] Peter G. Neumann. *Computer Related Risks*. Addison Wesley, Reading, Mass., 1995.

[7] T. Stabell-Kulø, F. Dillema, and T. Fallmyr. The open-end argument for private computing. In *Proceedings of the First International Symposium on Handheld and Ubiquitous Computing*, pages 124–136. Springer Verlag, Germany, 1999.