Pervasive computing mandates interaction between autonomous, mobile and possibly hostile entities. Ensuring that the interacting entity is trustworthy becomes a serious problem. This is, in addition to the problems like eavesdropping where, a malicious user listens on a network for information that might be useful. Unless the pervasive computing environment can be guaranteed to be secure and reliable, it will not be half as useful and profound as it promises to be.

When an entity interacts with another, there needs to be trust between them. There must be a minimum level of confidence that information sent out is not subject to misuse. A classic example is that of intermediate nodes in ad hoc networks. While forwarding packets, the nodes simply assume that the receiving node is trusty. However, the concept of trust is very subjective. So it becomes even more difficult to implement it in a software realm. Simply put, trust can be defined as the probability that an entity will behave as expected. This implies that there is always a risk associated with trust. Trust levels may vary among entities. Two entities need not trust each other equally. Trust may be inferred from personal knowledge about the entity or recommendations from trusted third parties.

In a software implementation, trust can be computed from the risk-PDFs (Probability Density Function) to a reasonably accurate level.

While trust is a subjective concept and has implementation problems associated, password authentication mechanisms have problems of their own. Current systems authenticate a user once and assume that the user’s identity does not change over time. They are based on a persistent authentication mechanism. This may lead to security threats in case a user leaves a workstation or a computing device unattended for some time. To overcome this problem, a transient authentication mechanism has been proposed that authenticates the identity of the user frequently. Since most users would not be willing to use the system if they had to do this manually, a token does this work on behalf of the user. This token has to be carried around or worn by the user at all times.

The system continuously polls the token that authenticates the identity of the user. When the user leaves the system unattended for a certain period of time, the poll fails and the system goes into hibernation blocking access. The tasks that are running on it are preserved in the same state by swapping the pages in memory. When the user returns, the system polls the token to find that the user has returned and resumes operation by restoring its state. To the user this operation is transparent and it appears as if nothing has
changed since he left the system. The problem with this implementation comes in implementing the hibernation of processes running. It may be the case that a certain process is in uninterruptible state and continues to run until it’s finished. The ratio of memory size to processing power needs to be reasonably small. If not, this would slow down the restoration process and the system would be less usable. To ensure sustained usability four criteria are stated that the system must meet:

1. The ability to perform sensitive operations must reside with the user
2. The process to secure and unlock the system needs to be only faster than the people using it
3. The securing process must not incur any additional burden either on the system or the user
4. User’s explicit content must be sought to carry out secure operations

Another problem though it may appear insignificant, is loss of tokens. The token typically would be a PDA or a smart card that a user carries all the time. If this is lost then the system would become inaccessible. Stolen tokens are another problem. The suggested mechanism of authenticating tokens once every few days does not offer any solution to the problem of stolen tokens or misplaced tokens that might be used by malicious users in the absence of the authentic user.

Security of systems is also of concern in public internet terminals. With software that can read keystrokes, there is always a risk of stolen passwords. To address this issue, a photographic authentication mechanism has been proposed. A user is required to upload his photographs to a trusted “home server”. When a user needs to access a secure web service, he first connects to his home server that presents a set of randomly selected photographs among which the user is required to identify his own. On correctly doing so, his credentials are passed on to the web service desired by the home server. Since this way, the credentials are passed by the home server, there is no risk of losing passwords. Also since the photographs can be identified by a mouse click and a different set is presented each time the user connects, chances of breaking this security are negligible.

The only possible threat to this system is from close acquaintances of the user who may be equally good at identifying the photographs of the user. This kind of threat is assumed to be non-existent. To some extent time-outs can be implemented wherein the system
times out if the user is unable to identify his photograph. If such attempts repeatedly occur, then the system is possible in the hands of a malicious user and the account can be locked preventing any further attempts.