Modern Security with Traditional Distributed Algorithms

Rachid Guerraoui and Marko Vukolić

Distributed Programming Laboratory, EPFL
CH 1015, Lausanne, Switzerland

Security modules and omission failures

Several manufacturers have recently started to equip their hardware with *security modules*. These typically consist of smart cards or special microprocessors. Examples include the "Embedded Security Subsystem" within the recent IBM Thinkpad or the IBM 4758 secure co-processor board [4]. In fact, a large body of computer and device manufacturers has founded the Trusted Computing Group (TCG) [9] to promote this idea.

In short, the computer hosts, besides its regular processor that can potentially be controlled by a malicious user, a trusted security module (Fig. 1). Because its hardware is tamper proof, the software running within a security module is certified and security modules can communicate through secure channels. However, communication goes through the untrusted hosts and dishonest ones can drop messages exchanged between the underlying security modules. As a consequence, the security modules form a distributed system of processes that can suffer from *general omission failures* [7] (i.e., either send or receive omission failures).

In other words, the very existence of security modules transforms malicious behavior into omissions. These omissions are not however random but can be committed by dishonest hosts at specific points of the computation.

In the following, we illustrate the transformation and some of the underlying issues through the problem of *multi-party fair exchange.* This problem is key to trading electronic items in systems of mutually untrusted parties. Each party expects to trade an item for another one, and each item has a description that is supposed to match this item. Each party hosts a security module and we assume here a synchronous model of computation, i.e., communication between security modules is synchronous and secure [3, 6], yet omissions can be committed.

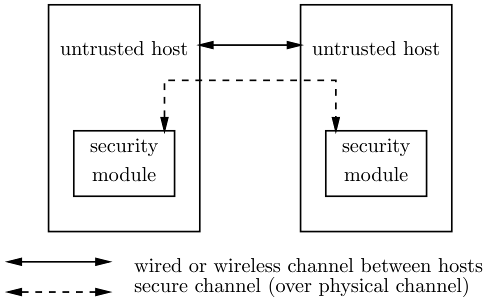

Fig. 1. Hosts and security modules.

Fair exchange and biased consensus

Definition 1 (Fair Exchange). *An algorithm solves* fair exchange *(FE) if it satisfies the following properties:*

- *(Timeliness) Every honest party eventually obtains the desired item or aborts the exchange.*
- *(Effectiveness) If no party misbehaves and all items match their descriptions then, upon termination, every party obtains the expected item.*
- *(Fairness) If any item does not match its description, or any honest party does not obtain its expected item, then no party obtains (any useful information about) any other party's item.*

Interestingly, solving fair exchange at the level of untrusted hosts is, in a precise sense, equivalent to solving the following variant of binary consensus [1] at the level of security modules, called processes here, and assuming that these can fail by omissions (correct processes are those that do not fail). We call this variant of binary consensus, *biased consensus* (BC), and we define it through the following properties:

Definition 2 (Biased Consensus).

- *(Termination) Every correct process eventually decides (0 or 1).*
- *(Non-Triviality) If no process is faulty or proposes 0, then no correct process decides 0.*
- *(Validity) No process decides 1 if some process proposes 0.*
- *(Biased Agreement) If any process decides 1, then no correct process decides 0.*

Biased consensus (BC) is different from consensus in the sense that (a) 0 can be decided even if all processes propose 1, and (b) some process might decide 0 whereas others might decide 1. In the first sense (a), BC is closer to non-blocking atomic commit (NBAC) [8]: in the second sense (i.e., b), they are however different. Both NBAC and BC are instances of *weak consensus* [5].

We give in [1] an algorithm that solves BC (and hence leads to solve fair exchange) assuming a synchronous system and a majority of correct processes. The algorithm is early stopping in the sense that the number of communication rounds needed to decide and terminate the algorithm depends on the number of actual failures. If there is no failure, two rounds are enough and this is clearly optimal. [1]

Massive attacks and probabilistic fairness

BC is not solvable if there is no correct majority of processes, i.e. if half or more processes can fail. This result, which we establish in [1], implies that FE is not solvable without a majority of honest parties either, even in a synchronous system with tamper proof modules. This motivates the study of a weaker variant of FE that could tolerate an arbitrary number of dishonest parties. We introduce the following problem, which we call *gracefully degrading fair exchange*, as a viable alternative to FE.

Definition 3 (Gracefully Degrading Fair Exchange). *An algorithm solves* gracefully degrading fair exchange *(GDFE) if it satisfies the following properties:*

- *The algorithm always satisfies the Timeliness and Effectiveness properties of fair exchange.*

---

[1] The proof is a simple variant of [5].

- *If a majority of parties are honest, then the algorithm also satisfies the Fairness property of fair exchange.*
- *Otherwise (if there is no honest majority), the algorithm satisfies Fairness with a probability* p *(*$0 < p < 1$*) such that the probability of unfairness (*$1 - p$*) can be made arbitrarily low.*

It is important to notice that the problem we introduce above trades *fairness* to cope with massive attacks (the case where at least half are dishonest). One could wonder why, as in randomized consensus [2], we have not chosen to trade *termination* instead. In fact, this would have led to exactly the same problem: not terminating means not obtaining any item and precisely means unfairness if some parties obtained their desired items. It is not however clear how *effectiveness* could be weakened in a non-trivial way and yet tolerate an arbitrary number of dishonest parties.

Intuitively, GDFE ensures that dishonest parties can only violate *fairness* by accident. We give a modular solution to GDFE in [1] which uses the early stopping BC algorithm discussed above as an underlying building block. Basically, the security modules first exchange a random number that indicates when (i.e., after how many communication rounds) BC is supposed to start and perform the actual exchange of items. Until that point, the security modules go through a series of *fake* communication rounds that the dishonest parties cannot distinguish from the actual BC communication pattern. Any omission at this stage simply leads to aborting the exchange.[2] The fact that we make use of an early stopping BC algorithm diminishes the probability for the dishonest parties to provoke omissions in such a way that they violate *fairness* in their favor. Unfairness of our algorithm is inversely proportional to its complexity. More precisely, considering a bi-uniform probability distribution [1], we show that the probability of violating *fairness* is in the order of $U_{GDFE} \approx 2/N$, where N is the upper bound on the range from which the random number of rounds is chosen from.

In fact, we can derive from [10] the fact that no GDFE algorithm, with maximal possible number of rounds N, can have a probability of unfairness that is less than $1/N$. The presence of the number 2 in our case might be intuitively explained by the very fact that we ensure deterministic fairness with a majority of honest parties (whereas [10] does not). Hence, at least two rounds of any GDFE algorithm are vulnerable. Proving that $2/N$ is optimal remains to be formally shown.

Applying our approach to non-synchronous systems as well as to other problems (i.e., besides fair exchange) opens interesting research directions.

Acknowledgments

This position paper is based on joint work with Gildas Avoine and Felix Gaertner.

---

[2] Dishonest parties might know the algorithm but not the random number, which is chosen at every execution and kept secret among security modules.

References

1. G. Avoine, F. Gärtner, R. Guerraoui, and M. Vukolić. Gracefully degrading fair exchange with security modules. Technical Report IC/2004/26, EPFL, Switzerland, 2004.
2. M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proc. Second Ann. ACM Symp. on Principles of Distributed Computing*, pages 27–30, 1983.
3. M. Correia, P. Veríssimo, and N. F. Neves. The design of a COTS real-time distributed security kernel. In *Proc. of the Fourth European Dependable Computing Conference*, Toulouse, France, Oct. 2002.
4. J. G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S. W. Smith, and S. Weingart. Building the IBM 4758 secure coprocessor. *IEEE Computer*, 34(10):57–66, Oct. 2001.
5. I. Keidar and S. Rajsbaum. On the cost of fault-tolerant consensus when there are no faults - a tutorial. Technical Report MIT-LCS-TR-821, MIT, May 24, 2001.
6. N. Lynch. *Distributed Algorithms.* Morgan Kaufmann, San Mateo, CA, 1996.
7. K. J. Perry and S. Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, 12(3):477–482, Mar. 1986.
8. D. Skeen. Non-blocking commit protocols. In *Proc. ACM SIGMOD Conf.*, page 133, Ann Arbor, MI, Apr.-May 1981.
9. Trusted Computing Group. Trusted computing group homepage. Internet: `https://www.trustedcomputinggroup.org/`, 2003.
10. G. Varghese and N. A. Lynch. A tradeoff between safety and liveness for randomized coordinated attack. *Information and Computation*, 128(1):57–71, 1996.