Batch Verification
with Applications to Cryptography and Checking

(Invited Paper)

Mihir Bellare¹*, Juan A. Garay², and Tal Rabin²

¹ Department of Computer Science & Engineering, Mail Code 0114,
University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA.
mhir@cs.ucsd.edu http://www-cse.ucsd.edu/users/mihir

² IBM T.J. Watson Research Center,
PO Box 704, Yorktown Heights, New York 10598, USA.

Abstract. Let \( R(\cdot) \) be a polynomial time-computable boolean relation.
Suppose we are given a sequence \( \text{inst}_1, \ldots, \text{inst}_n \) of instances and asked
whether it is the case that \( R(\text{inst}_i) = 1 \) for all \( i = 1, \ldots, n \). The naive
way to figure out the answer is to compute \( R(\text{inst}_i) \) for each \( i \) and check
that we get 1 each time. But this takes \( n \) computations of \( R \). Can one
do any better?
The above is the "batch verification" problem. We initiate a broad in-
estigation of it. We look at the possibility of designing probabilistic
batch verifiers, or tests, for basic mathematical relations \( R \). Our main
results are for modular exponentiation, an expensive operation in terms
of number of multiplications: here \( g \) is some fixed element of a group
\( G \) and \( R(x, y) = 1 \) iff \( g^x = y \). We find surprisingly fast batch verifiers
for this relation. We also find efficient batch verifiers for the degrees of
polynomials.
The first application is to cryptography, where modular exponentiation
is a common component of a large number of protocols, including digital
signatures, bit commitment, and zero knowledge. Similarly, the prob-
lem of verifying the degrees of polynomials underlies (verifiable) secret
sharing, which in turn underlies many secure distributed protocols.
The second application is to program checking. We can use batch verifi-
cation to provide faster batch checkers, in the sense of [20], for modular
exponentiation. These checkers also have stronger properties than stan-
dard ones, and illustrate how batch verification can not only speed up
how we do old things, but also enable us to do new things.

* Work supported in part by NSF CAREER Award CCR-9624439 and a 1996 Packard
Foundation Fellowship in Science and Engineering.
1 Introduction

We suggest the notion of batch verification. Based on this we suggest and implement a new paradigm for program checking [7]. We also suggest applications in cryptography. Motivated by this we design batch verifiers for some particular functions of interest in these domains.

1.1 Batch Verification

Let $R$ be a (polynomial time-computable, boolean) relation. The verification problem for $R$ is given an instance $\text{inst}$, check whether $R(\text{inst}) = 1$. In the batch verification problem we are given a sequence $\text{inst}_1, \ldots, \text{inst}_n$ of instances and asked to verify that for all $i = 1, \ldots, n$ we have $R(\text{inst}_i) = 1$. The naive way is to compute $R(\text{inst}_i)$, and check it is 1, for all $i = 1, \ldots, n$. We want to do it faster. To do this, we allow probabilism and an error probability. A batch verifier (also called a test) is a probabilistic algorithm $V$ which takes $\text{inst}_1, \ldots, \text{inst}_n$ and produces a bit as output. We ask that when $R(\text{inst}_i) = 1$ for all $i = 1, \ldots, n$, this output be 1. On the other hand, if there is even a single $i$ for which $R(\text{inst}_i) = 0$ then we want that $V(\text{inst}_1, \ldots, \text{inst}_n) = 1$ with very low probability. Specifically, we let $l$ be a security parameter and ask that this probability be at most $2^{-l}$.

We stress that if even a single one of the $n$ instances is "wrong" the verifier should detect it, except with probability $2^{-l}$. Yet we want this verifier to run faster than the time to do $n$ computations of $R$.

1.2 Application Domains

Batch verification will be useful in any algorithmic setting where there are repetitive tasks. Before presenting our results and the particular applications that ensue, let us briefly discuss two concrete application domains that have motivated our work.

Cryptography. It is a consequence of the "adversarial" nature of cryptography that many of its computational tasks are for the purpose of "verifying" some property or computation. A setting where batch verification is useful is in the verification of digital signatures. For example, the validity of a sequence of electronic coins needs to be verified by checking the bank's signature on each coin. When there are lots of coins, batch verification will help. Similarly one may receive many certificates, containing public keys signed by a certification authority, and one can check all the signatures simultaneously.

Beyond this, batch verification is useful for a large number of standard cryptographic protocols. These protocols typically involve repetition of some operation, such as a committal, done for example via the discrete exponentiation function $x \mapsto g^x$ in a group with generator $g$, so that a party commits to $x$ by providing $y = g^x$, and later de-commits by revealing $x$. At this point, someone must check that indeed $y = g^x$. In a zero-knowledge protocol, thousands or more committals are being performed simultaneously, and batch verification will be useful. The