Proving You Can Pick Stocks Without Revealing How (Internet Appendix)

Alex Chinco*

July 2, 2026
[Click here for the latest version]

A Encryption Schemes

This appendix describes three different homomorphic encryption schemes. The first is partially homomorphic. The second two are fully homomorphic.

A.1 Paillier

Paillier (1999) gives a public-key encryption scheme that can add pairs of ciphertexts. It cannot multiply two encrypted numbers and recover the product. This puts Paillier in the partially homomorphic family. Its security rests on the Decisional Composite Residuosity assumption, the conjecture that mth residues are hard to tell apart from non-residues modulo m^2. Inside the `EncryptIt` protocol, Paillier is the scheme for the case where Alice has no stock-picking skill of her own.

How the scheme works. The scheme is built around a modulus made from two large primes. To create her secret key, Alice first picks two large prime numbers, a and b. Let $m = a \cdot b$ denote the product of the two chosen primes. Alice's secret key is the least common multiple of the two even numbers immediately below each prime, $sk = \text{LCM}(a-1, b-1)$. The public key is the modulus itself, $pk = m$.

The message is a non-negative integer less than m, $msg \in \{0, 1, \ldots, (m-1)\}$. To encrypt his choice, the manager first picks a fresh random number $\varepsilon \sim$

*Michigan State University. alexchinco@gmail.com

$\{0, 1, \ldots, (m-1)\}$ that shares no common divisors with m. He then forms the ciphertext

$$\texttt{Enc}(msg) \;=\; \left[\, (m+1)^{msg} \cdot \varepsilon^{m} \,\right] \bmod m^2 \tag{A.1}$$

The manager's random choice of ε makes the scheme probabilistic. Encrypt the same message twice and you get two unrelated-looking ciphertexts.

To decrypt the manager's ciphertext, Alice calculates

$$\left[\, (sk^{-1} \bmod m) \cdot \frac{(\texttt{Enc}(msg)^{\,sk} \bmod m^2) - 1}{m} \,\right] \bmod m \tag{A.2}$$

The end result is $\texttt{Dec}[\texttt{Enc}(msg)] = msg$.

Example ciphertext. Suppose Alice picks primes $a = 61$ and $b = 53$, which gives $m = 61 \cdot 53 = 3{,}233$ and $m^2 = 10{,}452{,}289$. These choices give a secret key of $sk = \text{LCM}(60, 52) = 780$ and a public key of $pk = m = 3{,}233$.

Suppose a manager wants to encrypt a single buy recommendation, \$0.50. This is not an integer, so let us use $msg = 50$ instead. Before encrypting, the manager draws a value at random from 0 to 3,232. Suppose he picks $\varepsilon = 1{,}234$. The resulting Paillier ciphertext is

$$\texttt{Enc}(50) \;=\; \left[\, (3{,}233+1)^{50} \cdot 1{,}234^{3{,}233} \,\right] \bmod 10{,}452{,}289 \;=\; 7{,}374{,}277 \tag{A.3}$$

A Paillier ciphertext is a single integer, the same kind of object as the message but at a much larger scale.

Real deployments use a modulus m of at least 2,048 bits. Squaring doubles the bit length, so m^2 runs to about 4,096 bits. That puts the length of each ciphertext at length$(m^2) = \log_{10}(2) \times 4{,}096 \approx 1{,}200$ decimal digits.

Homomorphic structure. The encrypted-addition operation, $\oplus$, is just multiplication modulo m^2

$$\begin{aligned} \texttt{Enc}(msg) \,\oplus\, \texttt{Enc}(msg') \;&=\; \left[\, \texttt{Enc}(msg) \cdot \texttt{Enc}(msg') \,\right] \bmod m^2 && \text{(A.4a)} \\ &=\; \texttt{Enc}\big(\, [msg + msg'] \bmod m \,\big) && \text{(A.4b)} \end{aligned}$$

This works because the message rides in the exponent of $(m+1)$. When you multiply powers, the exponents add, $x^3 \cdot x^2 = x^{3+2} = x^5$.

A second property follows the same way. Raise a ciphertext to a known integer power z and it is like multiplying the message by z

$$\texttt{Enc}(msg)^{z} \bmod m^2 \;=\; \texttt{Enc}\big(\, [z \cdot msg] \bmod m \,\big) \tag{A.5}$$

So Paillier can add pairs of encrypted messages, $\oplus$, and it can multiply an encrypted message by a plaintext constant. But this is not $\otimes$. There is no way to multiply a pair of encrypted numbers in Paillier.

The EncryptIt protocol. Paillier implements `EncryptIt` only under the assumption that Alice has no stock-picking skill of her own. Take two stocks. The manager encrypts the two positions $Position_1$ and $Position_2$. The notary raises each ciphertext to the power of the matching realized return, $Return_1$ and $Return_2$, and combines the two with $\oplus$

$$\begin{aligned} &\texttt{Enc}(Position_1)^{Return_1} \oplus \texttt{Enc}(Position_2)^{Return_2} \\ &\qquad = \texttt{Enc}\big(\, Position_1 \cdot Return_1 + Position_2 \cdot Return_2 \,\big) && \text{(A.6a)} \\ &\qquad = \texttt{Enc}(\pi) && \text{(A.6b)} \end{aligned}$$

A negative return enters as a negative exponent, which is just a modular inverse. A negative position is handled on the message side. A Paillier message must lie between 0 and $(m-1)$, so the long position +50 is already valid while the short position -50 is represented by its residue modulo m, the message $(m-50) = 3{,}183$. Alice holds the secret key and decrypts $\texttt{Enc}(\pi)$ to read the profit. The toy numbers carry through. The two positions encode to the messages 50 and 3,183, which encrypt to 7,374,277 and 9,534,319. With public returns +1 and -1, the notary forms an encrypted profit of 164,786, which decrypts to 100, exactly $50 \cdot 1 + (-50) \cdot (-1)$.

This is fine when the returns are the raw market returns. It fails if Alice hands the notary $\texttt{Enc}(Resid_1)$ and $\texttt{Enc}(Resid_2)$ rather than the plaintext residuals. The notary would then need $\otimes$, multiplying $\texttt{Enc}(Position_1)$ by $\texttt{Enc}(Resid_1)$, and Paillier has no such operation. So Paillier runs the protocol exactly when Alice has no stock-picking skill of her own, and not otherwise.

A.2 van Dijk, Gentry, Halevi, and Vaikuntanathan

van Dijk, Gentry, Halevi, and Vaikuntanathan (2010, DGHV) built a fully homomorphic encryption scheme out of integer arithmetic. The authors developed the approach the year after Craig Gentry introduced the first fully homomorphic scheme (Gentry, 2009). The goal was to make Gentry's idea as simple as possible. It is a proof of concept rather than a practical encryption scheme. The message is a single bit. Its ciphertext is a single integer. Unlike Paillier, DGHV is not limited to just $\oplus$. It can do $\otimes$ as well. Both operations are done modulo 2. Because DGHV can multiply two encrypted numbers, it can be used to implement `EncryptIt` even when Alice has stock-picking skill of her own. DGHV's security guarantee rests on the approximate-GCD problem.

How the scheme works. The message in DGHV is a single bit, $msg \in \{0, 1\}$. Longer messages are dealt with bitwise. More on that later. To create her secret key, sk, Alice chooses a large positive odd integer.

Generating the public key is more complicated. It is a two-step process. The first step involves creating a list of $K \gg 1$ ciphertexts that all decode to zero. For each entry, Alice must randomly draw two objects, a multiplier, m_k, and a small noise term, ξ_k. The kth ciphertext is $sk \cdot m_k + 2 \cdot \xi_k$. Each of the K ciphertexts is just a number. In the second step, Alice computes the sum for all possible subsets. The resulting vector with $(2^K - 1)$ entries constitutes the public key. For example, if we set $K = 2$, then

$$pk = \{\ (sk \cdot m_1 + 2 \cdot \xi_1),\quad (sk \cdot m_2 + 2 \cdot \xi_2),\quad (sk \cdot [m_1 + m_2] + 2 \cdot [\xi_1 + \xi_2])\ \} \tag{A.7}$$

Without the small noise term, ξ_k, these ciphertexts would be exact multiples of sk. The greatest common divisor of any two of them would reveal Alice's secret key. Anyone would be able to reverse-engineer the secret key. The small noise term makes each ciphertext a near-multiple. As a result, cracking the encryption scheme is as hard as approximate-GCD.

To encrypt a message, the manager adds the bit to a new small noise term, ε, and picks an entry at random from the public key. I will use $\langle m \rangle$ and $\langle \xi \rangle$ to denote the sum over multipliers and noise terms in the manager's selected entry from the public key. Putting the pieces together, we get

$$\begin{aligned} \texttt{Enc}(msg) &= msg + 2 \cdot \varepsilon + (sk \cdot \langle m \rangle + 2 \cdot \langle \xi \rangle) && \text{(A.8a)} \\ &= msg + sk \cdot \langle m \rangle + 2 \cdot (\varepsilon + \langle \xi \rangle) && \text{(A.8b)} \end{aligned}$$

To decrypt a ciphertext, Alice first reduces modulo sk and then modulo 2,

$$[\ \texttt{Enc}(msg) \bmod sk\ ] \bmod 2 \tag{A.9}$$

which returns the message, $\texttt{Dec}[\texttt{Enc}(msg)] = msg$. Reducing by sk wipes out the term that includes her secret key, leaving $msg + 2 \cdot (\varepsilon + \langle \xi \rangle)$. Reducing by 2 kills off the second term, leaving just the original message.

Example ciphertext. A DGHV ciphertext is a single integer. Suppose Alice picks $sk = 2{,}773$ for her secret key. For simplicity, let us take $K = 2$. Suppose Alice picks multipliers $m_1 = 3{,}517$ and $m_2 = 1{,}448$. Her two small noise terms are $\xi_1 = 3$ and $\xi_2 = 1$. This gives her the following pair of ciphertexts

$$\begin{aligned} \overset{sk}{2{,}773} \cdot \overset{m_k}{3{,}517} + 2 \cdot \overset{\xi_k}{3} &= 9{,}752{,}647 && \text{(A.10a)} \\ 2{,}773 \cdot 1{,}448 + 2 \cdot 1 &= 4{,}015{,}306 && \text{(A.10b)} \end{aligned}$$

Alice's public key would then be

$$pk = \{\ 9{,}752{,}647,\quad 4{,}015{,}306,\quad 13{,}767{,}953\ \} \tag{A.11}$$

DGHV deals with single-bit messages. So let us write a buy position as $msg = 1$. To encrypt this message, the manager first adds a small noise term, $\varepsilon = 2$, to his message. Then he randomly picks one of the entries from the public key, say 13,767,953. The manager adds this amount to get the final encrypted message

$$\mathtt{Enc}(1) \;=\; 1 \;+\; \overset{\varepsilon}{2} \cdot \overset{sk\cdot\langle m\rangle + 2\cdot\langle \xi\rangle}{2} \;+\; 13{,}767{,}953 \;=\; 13{,}767{,}958 \tag{A.12}$$

To decrypt the message, Alice first reduces by sk to get 13,767,958 mod 2,773 = 13. Then she reduces again by 2 to recover 13 mod 2 = 1.

Homomorphic structure. For DGHV, the two homomorphic operations, $\oplus$ and $\otimes$, are nothing more than ordinary integer addition and multiplication, + and $\times$. It just so happens that the pair of operations can be transferred inside the $\mathtt{Enc}(\cdot)$ operator. Adding two ciphertexts works as follows

$$\begin{aligned} \mathtt{Enc}(msg) \,\oplus\, \mathtt{Enc}(msg') \;&=\; \mathtt{Enc}(msg) + \mathtt{Enc}(msg') && \text{(A.13a)} \\ &=\; \mathtt{Enc}\big(\,[msg + msg'] \bmod 2\,\big) && \text{(A.13b)} \end{aligned}$$

Multiplying two ciphertexts is done in a similar way

$$\begin{aligned} \mathtt{Enc}(msg) \,\otimes\, \mathtt{Enc}(msg') \;&=\; \mathtt{Enc}(msg) \times \mathtt{Enc}(msg') && \text{(A.14a)} \\ &=\; \mathtt{Enc}\big(\,[msg \times msg'] \bmod 2\,\big) && \text{(A.14b)} \end{aligned}$$

The small even term left after reducing by sk is noise. The size of this term gets larger after each addition and multiplication. Once the noise passes $sk/2$, the reduction modulo sk wraps around and decryption fails. Thus only a bounded number of operations are safe before the ciphertext must be refreshed.

The EncryptIt protocol. Because DGHV supplies $\otimes$ as well as $\oplus$, it can be used to implement `EncryptIt` in full. Alice can encrypt her own residual returns and hand the notary $\mathtt{Enc}(Resid_1)$ and $\mathtt{Enc}(Resid_2)$, and the notary can still form the encrypted profit, because multiplying an encrypted position by an encrypted residual is now an allowed operation.

The work is all done bit by bit. Each position and each return is written in binary, and every bit is a separate ciphertext. To handle the carry operations that occur when adding and multiplying large numbers, DGHV models the inner product $Position_1 \cdot Return_1 + Position_2 \cdot Return_2$ as a boolean circuit of XOR and AND gates over those encrypted bits. This is complete overkill, like showing up with a bazooka to a thumb war. You do not need a Turing-complete computing environment to carry the one in 5 + 6 = 11.

DGHV shows that the general protocol is possible with nothing more than integer addition and multiplication. It does not show that it is practical.

A.3 Brakerski and Vaikuntanathan

Brakerski and Vaikuntanathan (2011, BV) follows a similar approach to DGHV. Both bury the message in noise and a secret mask, and both dig it back out with two reductions. The difference is in how the computational steps are handled on the backend. DGHV handles the "carry" operations that occur when adding and multiplying numbers by creating an AND-XOR circuit. This is complete overkill. You do not need such a sophisticated machine to calculate $5 + 6 = 11$. BV cuts back on all this by encrypting the message in the coefficients of a polynomial ring. The security rests on ring-LWE, the hardness of recovering a small secret polynomial from noisy products. It supplies both $\oplus$ and $\otimes$, so like DGHV it runs `EncryptIt` even when Alice has stock-picking skill of her own, but it gets there far more cheaply.

How the scheme works. DGHV worked over the integers. BV uses polynomials. Fix the ring $\mathbf{Z}_Q[x]/(x^D+1)$, which involves polynomials of degree $(D-1)$ with integer coefficients chosen from $\{0, 1, \ldots, (Q-1)\}$. For example, if $D = 4$, then the ring is over polynomials with x^3, x^2, x^1, and x^0.

A polynomial here is just a way to write a number. The placeholder x plays the role the 10 plays in base ten, and the coefficients are the digits. We could write the number 273 as $0 \cdot 10^3 + 2 \cdot 10^2 + 7 \cdot 10^1 + 3 \cdot 10^0$. A message is a number with digits chosen from a much smaller list, $\{0, 1, \ldots, (q-1)\}$ where $q \ll Q$.

D, Q, and q are all public knowledge. Each input bounds a different thing. D puts a cap on the size of the message that can be encrypted. Q is the noise budget, the room the overshoot has before it reaches $Q/2$ and decryption fails. The noise comes in units of q. This means that we need $q \ll Q$. The ratio $Q/q \gg 1$ determines how deep a computation can run.

The secret key is a column vector where s is a random polynomial with small coefficients

$$sk = \begin{pmatrix} 1 \\ s \end{pmatrix} \tag{A.15}$$

The public key in BV is the column vector

$$pk = \begin{pmatrix} p_0 \\ p_1 \end{pmatrix} = \begin{pmatrix} -q \cdot \xi & -m \\ m & 0 \end{pmatrix} \begin{pmatrix} 1 \\ s \end{pmatrix} = \begin{pmatrix} -q \cdot \xi - m \cdot s \\ m \end{pmatrix} \tag{A.16}$$

m be a random polynomial and ξ is a small noise term. What Alice publishes is the 2×1 vector on the right. m is public information, so the small noise term ξ is needed to prevent p_0 from revealing the secret polynomial, s. If the public key were defined as $m \cdot \binom{-s}{1}$, then the secret key would not be secret.

Multiplying the secret and public keys returns a small error

$$sk^\top pk = p_0 + p_1 \cdot s \tag{A.17a}$$
$$= -q \cdot \xi - m \cdot s + m \cdot s = -q \cdot \xi \tag{A.17b}$$

The positive and negative $m \cdot s$ terms cancel. All that remains is $-(q \cdot \xi)$, and this term reduces to zero modulo q.

To encrypt a message, the manager draws two new random objects. The first is a small polynomial, u. The second is a (2×1)-dimensional vector of polynomials, $\varepsilon = \binom{\varepsilon_0}{\varepsilon_1}$. He then computes the following ciphertext

$$\texttt{Enc}(msg) = u \cdot \begin{pmatrix} p_0 \\ p_1 \end{pmatrix} + q \cdot \begin{pmatrix} \varepsilon_0 \\ \varepsilon_1 \end{pmatrix} + msg \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{A.18}$$

u and ε serve different functions. The manager draws u fresh and keeps it secret. It hides his message by folding a secret multiple of the public key into the ciphertext. An outside observer cannot reproduce this step using the public key alone. The error ε hides u. Without it, the lower coordinate in the encrypted message would be $u \cdot m$ exactly since $p_1 = m$.

Decryption is a multi-step process. Alice starts by left-multiplying by her secret key. After that, she reduces twice: first modulo Q and then modulo q

$$\texttt{Dec}[\texttt{Enc}(msg)] = \Big(\big[\, sk^\top \texttt{Enc}(msg) \,\big] \bmod Q \Big) \bmod q = msg \tag{A.19}$$

It is the same approach as DGHV, with the plaintext modulus q in place of 2.

Left-multiplying by the secret key transforms the 2×1 vector-valued ciphertext into a single polynomial

$$sk^\top \texttt{Enc}(msg) = u \cdot \begin{pmatrix} 1 & s \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \end{pmatrix} + q \cdot \begin{pmatrix} 1 & s \end{pmatrix} \begin{pmatrix} \varepsilon_0 \\ \varepsilon_1 \end{pmatrix} + msg \tag{A.20a}$$
$$= u \cdot (sk^\top pk) + q \cdot (sk^\top \varepsilon) + msg \tag{A.20b}$$

Reducing modulo Q leaves the message unaffected so long as the noise stays below $Q/2$. Reducing modulo q strips the noise away.

Example ciphertext. Take $D = 4$, which gives polynomials of degree $(4-1) = 3$. Assume a coefficient modulus of $Q = 2{,}097{,}169$ and a plaintext modulus of $q = 29$. A buy position in the paper takes the value +\$0.50. We write this number as $50 = 5 \cdot 10^1 + 0 \cdot 10^0$. If we replace each 10 with an x, then we get the polynomial $msg = 0 \cdot x^3 + 0 \cdot x^2 + 5 \cdot x^1 + 0 \cdot x^0$.

Suppose Alice's secret polynomial is $s = -1 \cdot x^3 + 1 \cdot x^2 + 1 \cdot x^1 - 1 \cdot x^0$. To build the public key, Alice draws a random ring element, $m = 55{,}230 \cdot x^3 + 274{,}847 \cdot x^2 + 1{,}988{,}324 \cdot x + 1{,}551{,}704$. This is a polynomial whose four coefficients are picked uniformly modulo Q. Alice also randomly samples a small error term, $\xi = -1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 0 \cdot x^0$. She combines these inputs to produce the following public key

$$
\begin{aligned}
pk &= \begin{pmatrix} -q \cdot \xi & -m \\ m & 0 \end{pmatrix} \begin{pmatrix} 1 \\ s \end{pmatrix} && \text{(A.21a)} \\
&= \begin{pmatrix} -q \cdot \xi - m \cdot s \\ m \end{pmatrix} && \text{(A.21b)} \\
&= \begin{pmatrix} 1{,}440{,}961 \\ 55{,}230 \end{pmatrix} \cdot x^3 + \begin{pmatrix} 873{,}898 \\ 274{,}847 \end{pmatrix} \cdot x^2 \\
&\qquad + \begin{pmatrix} 217{,}003 \\ 1{,}988{,}324 \end{pmatrix} \cdot x^1 + \begin{pmatrix} 1{,}990{,}626 \\ 1{,}551{,}704 \end{pmatrix} \cdot x^0 && \text{(A.21c)}
\end{aligned}
$$

To encrypt the buy position, the manager randomly draws a small polynomial, $u = 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x^1 - 1 \cdot x^0$. He also randomly draws a (2×1)-dimensional noise vector with elements $\varepsilon_0 = 1 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$ and $\varepsilon_1 = -1 \cdot x^3 + 1 \cdot x^2 - 1 \cdot x^1 - 1 \cdot x^0$. Then, he creates the following ciphertext

$$
\begin{aligned}
\texttt{Enc}(50) &= u \cdot pk + q \cdot \varepsilon + (0 \cdot x^3 + 0 \cdot x^2 + 5 \cdot x^1 + 0 \cdot x^0) \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} && \text{(A.22a)} \\
&= u \cdot \begin{pmatrix} p_0 \\ p_1 \end{pmatrix} + q \cdot \begin{pmatrix} \varepsilon_0 \\ \varepsilon_1 \end{pmatrix} + \begin{pmatrix} 5 \\ 0 \end{pmatrix} \cdot x^1 && \text{(A.22b)} \\
&= \begin{pmatrix} 1{,}423{,}592 \\ 1{,}771{,}292 \end{pmatrix} \cdot x^3 + \begin{pmatrix} 2{,}096{,}482 \\ 1{,}658{,}276 \end{pmatrix} \cdot x^2 \\
&\qquad + \begin{pmatrix} 899{,}730 \\ 1{,}385{,}673 \end{pmatrix} \cdot x^1 + \begin{pmatrix} 545{,}777 \\ 599{,}051 \end{pmatrix} \cdot x^0 && \text{(A.22c)}
\end{aligned}
$$

Alice decrypts this object in stages. She starts by left-multiplying with her secret key, $sk^\top \texttt{Enc}(50)$. Then, she reduces modulo Q and modulo q. The end result is the polynomial $0 \cdot x^3 + 0 \cdot x^2 + 5 \cdot x^1 + 0 \cdot x^0$, which is a fancy way of writing 50 in mathese.

Homomorphic structure. In BV, $\oplus$ and $\otimes$ represent coordinate-wise addition and multiplication. It works a bit like adding and multiplying pairs of complex numbers, but with a twist.

Let's start with addition. It is easier. We can write an encrypted ciphertext as

$$\texttt{Enc}(msg) \;=\; \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} \;=\; \begin{pmatrix} u \cdot p_0 + q \cdot \varepsilon_0 + msg \\ u \cdot p_1 + q \cdot \varepsilon_1 + 0 \end{pmatrix} \tag{A.23}$$

The first left-multiplication step of Alice's decryption process will spit out $sk^\top \texttt{Enc}(msg) \;=\; c_0 + c_1 \cdot s$. It shouldn't matter whether Alice does this step before or after the addition. If she does so beforehand, then we get

$$\begin{aligned} \left[sk^\top \texttt{Enc}(msg)\right] + \left[sk^\top \texttt{Enc}(msg')\right] \;&=\; [c_0 + c_1 \cdot s] + [c'_0 + c'_1 \cdot s] && \text{(A.24a)} \\ &=\; (c_0{+}c'_0) \cdot s^0 + (c_1{+}c'_1) \cdot s^1 && \text{(A.24b)} \end{aligned}$$

If Alice added together a pair of encrypted buy positions like the one above, the resulting ciphertext would decrypt to $50 + 50 = 100$. After reducing, the message will come back written as $0 \cdot x^3 + 0 \cdot x^2 + 10 \cdot x^1 + 0 \cdot x^0$. The 10^1 basis has a coefficient of 10. There is no issue here because we have not yet committed to $x = 10$. Once we do so, the leading 1 carries to the 10^2 base. Where DGHV would have rippled a carry through its circuit on the spot, BV lets the digit sit and defers the carry to decode. It just requires q to be large enough. For base 10 and a single round of addition, we need $q > 9 + 9 = 18$.

Multiplication is more complicated. Again, it shouldn't matter whether Alice decrypts before or after multiplying. If she does so beforehand, she will find

$$\begin{aligned} \left[sk^\top \texttt{Enc}(msg)\right] \times \left[sk^\top \texttt{Enc}(msg')\right] \;&=\; [c_0 + c_1 \cdot s] \times [c'_0 + c'_1 \cdot s] && \text{(A.25a)} \\ &=\; s^0 \cdot (c_0 \cdot c'_0) \\ &\quad + s^1 \cdot (c_0 \cdot c'_1 + c_1 \cdot c'_0) && \text{(A.25b)} \\ &\quad + s^2 \cdot (c_1 \cdot c'_1) \end{aligned}$$

The right-hand side contains a new s^2 term. The product ciphertext is a triple, not a pair. The secret key that decrypts an ordinary ciphertext is now one entry too short. So Alice extends it

$$sk = \begin{pmatrix} 1 & s \end{pmatrix} \rightarrow \begin{pmatrix} 1 & s & s^2 \end{pmatrix} \tag{A.26}$$

Each multiplication adds another power of s to the key. The one above took it from $(1,s)$ to $(1,s,s^2)$. A second would bring in s^3, a third s^4, and so on. The powers cannot stack without limit. Every one is an element of the ring

$\mathbf{Z}_Q[x]/(x^D{+}1)$, which is only D coefficients wide, so past the $(D{-}1)$th power they stop being independent. The power s^D is already some combination of the lower ones. The secret key can accommodate s^{D-1} and no more.

Both operations decrypt correctly because the secret-key product, minus the message, is a multiple of q

$$\begin{aligned} \left[sk^\top \mathtt{Enc}(msg)\right] - msg &= u \cdot [sk^\top pk] + q \cdot [sk^\top \varepsilon] && \text{(A.27a)} \\ &= u \cdot [-q \cdot \xi] + q \cdot [sk^\top \varepsilon] && \text{(A.27b)} \\ &= \left([sk^\top \varepsilon] - u \cdot \xi\right) \cdot q && \text{(A.27c)} \end{aligned}$$

The factor of q explains why nothing can be learned from the secret key that shows up in this residual. After reducing by q, the whole term goes to zero. Reducing modulo Q rescales the coefficients. The step is innocuous so long as the message coordinates remain less than $Q/2$. Reducing modulo q comes second and deletes the residual. This is why $q \ll Q$. The noise comes in units of q, and every operation enlarges it, climbing in steps of q toward the $Q/2$ ceiling. The ratio of these two constants, $Q/q \gg 1$, puts a limit on how deep a computation can run before the modulo-Q step wraps.

The EncryptIt protocol. Because BV supplies $\otimes$ as well as $\oplus$, the notary can run `EncryptIt` in full, including the case where Alice encrypts her own residual returns. This is the same reach as DGHV, but BV gets there far more cheaply. Each position is one number, so one ciphertext, and each return is one number, so one ciphertext. The notary multiplies each encrypted position by its encrypted return, one $\otimes$ per stock, and adds the results with $\oplus$, forming the encrypted profit $Position_1 \cdot Return_1 + Position_2 \cdot Return_2$. For a market of N stocks that is N multiplications and N additions of ciphertexts, each a single ring operation. DGHV would have written every number out in bits and built the same inner product as a boolean circuit of millions of gates.

The inner product is a sum of products, each position times its return, so it is only one multiplication deep. The message degree grows once, staying well under the ceiling $(D{-}1)$, and the noise grows once, both within what D and Q are sized to hold, so there is no need for bootstrapping. Reverse it into a product of sums and BV would be unusable. A calculation that is N multiplications deep would blow past $(D{-}1)$ within a few steps. The shape of the inner product, a sum of products rather than a product of sums, is what keeps it cheap.

B Private Commitment

Step #1 of Protocol 3 asks you to encrypt the positions implied by your buy/sell recommendations and send the resulting ciphertexts to the notary. Nothing about that step stops you from encrypting a \$5.00 position instead of a \$0.50 position, or from encrypting ten long positions and zero short ones. Left unchecked, either move breaks the correspondence between `EncryptIt` and `TradeOnIt` that Proposition 4 depends on. I show here that the notary can rule out both possibilities using nothing more than the homomorphic operations, $\oplus$ and $\otimes$. No separate cryptographic machinery is required, and none of the extra computation falls on Alice.

Two conditions have to hold. First, every individual position must be sized correctly, $Position_{n,t} \in \{+\$0.50, -\$0.50\}$ for each stock n. Second, the collection of positions as a whole must form a genuine long/short portfolio, meaning the book is dollar neutral, $\sum_{n=1}^{N} Position_{n,t} = 0$. Neither condition implies the other. A trader could size every individual position correctly and still submit a portfolio with no short positions at all.

Checking 1 position. A position clears the sizing bar exactly when its square equals a fixed constant

$$Position_{n,t}^2 \;=\; (\pm\$0.50)^2 \;=\; 0.25 \tag{B.28}$$

Squaring is something the notary can already do. It is the same $\otimes$ operation he uses to build $\mathtt{Enc}(\pi_t)$ from $\mathtt{Enc}(Position_{n,t}) \otimes \mathtt{Enc}(Resid_{n,t})$ in Equation (18).

Using the public key alone, he computes

$$\begin{aligned} \big[\,\mathtt{Enc}(Position_{n,t}) \otimes \mathtt{Enc}(Position_{n,t})\,\big] \;\oplus\; \mathtt{Enc}(-0.25) \\ = \;\mathtt{Enc}\big(\,Position_{n,t}^2 - 0.25\,\big) \end{aligned} \tag{B.29}$$

This ciphertext will decrypt to zero exactly when stock n's position clears the bar. Decrypting this value is safe. Whenever the position is valid, $Position_{n,t}^2$ equals the same constant, 0.25, regardless of whether the position is long or short, so learning it reveals nothing about which side of the trade you took.

Checking N positions. Decrypting Equation (B.29) for every stock would ask Alice to perform N decryptions each period, on top of the one she already performs to recover π_t. The notary can collapse all N checks into a single number instead, using a randomized consistency check in the tradition of Freivalds (1979) and Schwartz (1980). To do this, the notary draws N public weights, $\{w_{1,t}, \ldots, w_{N,t}\}$, only after he has already received your encrypted

positions. He uses these randomly selected weights to combine the N stock-specific discrepancies from Equation (B.29) into one ciphertext

$$\texttt{Enc}(Z_t) \;=\; \bigoplus_{n=1}^{N} w_{n,t} \otimes \texttt{Enc}\big(\, \mathit{Position}_{n,t}^2 - 0.25 \,\big) \tag{B.30}$$

Because the weights are drawn only after your positions are locked in and can no longer change, you have no way to size a position incorrectly in a way built to cancel out under a weight vector you have not seen. If every position clears the sizing bar, then it should not matter what the weights turn out to be. Whatever the notary chooses, he will still calculate a $\texttt{Enc}(Z_t)$ that decrypts to zero, $\texttt{Dec}[\texttt{Enc}(Z_t)] = 0$. A weighted sum of zeros is zero. If one or two positions are incorrectly sized, then the notary will almost never find $Z_t = 0$ by chance.

Alice's only new task is to decrypt $\texttt{Enc}(Z_t)$ once per period and confirm it equals zero, alongside the profit ciphertext she was already decrypting. Decrypting Z_t is safe precisely because it always equals the same constant, 0, whenever you are telling the truth. It carries no information about which stocks you held long, which you held short, or anything else about your positions.

Enforcing long/short book. The second condition, that the portfolio is dollar neutral, needs even less machinery. The notary already has $\texttt{Enc}(\mathit{Position}_{n,t})$ for every stock from Step #1. He sums them directly,

$$\texttt{Enc}(S_t) \;=\; \bigoplus_{n=1}^{N} \texttt{Enc}(\mathit{Position}_{n,t}) \;=\; \texttt{Enc}\left(\sum_{n=1}^{N} \mathit{Position}_{n,t} \right) \tag{B.31}$$

No multiplication is required, only the same repeated $\oplus$ he already uses to build up a sum. A genuinely balanced book, exactly $N/2$ long positions and $N/2$ short positions at ±\$0.50 each, gives $S_t = 0$ exactly. This requires N to be even, which holds in both of the simulated environments in Online Appendix C. Alice decrypts $\texttt{Enc}(S_t)$ alongside $\texttt{Enc}(Z_t)$ and $\texttt{Enc}(\pi_t)$ and confirms it is exactly zero.

Amending the protocol. Both checks slot into Step #1 of Protocol 3 without changing anything else about it. Immediately after receiving $\{\texttt{Enc}(\mathit{Position}_{n,t})\}_{n=1}^{N}$, the notary computes $\texttt{Enc}(Z_t)$ using Equation (B.30) and $\texttt{Enc}(S_t)$ using Equation (B.31). He holds onto both and sends them to Alice bundled with $\texttt{Enc}(\pi_t)$ at the end of the period, so she still performs only one decryption per period, now covering three numbers instead of one. Alice proceeds to compute α_t only if $\texttt{Dec}[\texttt{Enc}(Z_t)] = \texttt{Dec}[\texttt{Enc}(S_t)] = 0$. Otherwise, she concludes that what she received was not a valid long/short portfolio and treats it as a failed proof.

C Simulation Details

This appendix describes the simulations used to create Figures 2 and 3 in the main text.

C.1 Numerical Precision

Figure 2 in the main text reports the output of two simulations, one per panel. Both follow the same pipeline. Draw a vector of positions and a vector of residual returns for a market with $N = 500$ stocks. Encrypt each stock-level value. Compute the encrypted profit as a homomorphic dot product, one $\otimes$ per stock followed by a chain of $\oplus$ operations. Decrypt the total and compare the implied accuracy, $\alpha_t = \texttt{Dec}[\texttt{Enc}(\pi_t)] \,/\, (\$0.01 \times N)$, to the true accuracy parameter, α. The left panel uses the binary environment from the main text, $Position_{n,t} \in \{\pm\$0.50\}$ and $Resid_{n,t} \in \{\pm 1\%\}$. The right panel replaces these values with jointly normal draws. I run 100 iterations at each of nine accuracy levels, $\alpha \in \{0.00, 0.02, \ldots, 0.16\}$, so each panel contains 900 dots.

Encryption scheme. Both simulations use the Brakerski-Fan-Vercauteren (BFV) scheme (Brakerski, 2012; Fan and Vercauteren, 2012), a production-grade descendant of the Brakerski and Vaikuntanathan (2011) scheme described in Online Appendix A.3. It encrypts integers rather than bits, and its security rests on the same ring-LWE problem. The simulations are written in Python using the `TenSEAL` bindings for Microsoft's SEAL library. The polynomial modulus degree is 8,192. The plaintext modulus is 1,032,193. The ciphertext modulus chain is the library default for 128-bit security at this ring dimension.

Two facts about BFV matter for numerical precision. First, a BFV plaintext is a signed integer, and arithmetic on ciphertexts is exact so long as every intermediate value stays below half the plaintext modulus, roughly ±516,000 at these settings. There is no rounding inside the encryption. Second, a single ciphertext packs $8{,}192/2 = 4{,}096$ integer slots. One ciphertext holds all $N = 500$ positions, and a second holds all $N = 500$ residual returns. Element-wise homomorphic multiplication fills the slots with stock-level products, and a rotate-and-sum reduction built from Galois keys collapses the slots into a single encrypted total. This executes the encrypted profit calculation from Equation (18) in two library calls.

Binary simulation. Each iteration draws a portfolio uniformly at random from the set of balanced books, exactly $N/2$ long positions and $N/2$ short positions. The residual returns start out as a perfect copy of the positions, $\text{Sign}(Resid_{n,t}) = \text{Sign}(Position_{n,t})$ for every stock. The simulation then flips

$\frac{1}{2} \cdot \left(N - \text{round}[\, N \cdot (\frac{1}{2} + \alpha)\,]\right)$ of the long-side residuals to −1% and the same number of short-side residuals to +1%. The flips are symmetric, so both balance conditions survive. Half of all positions remain long, and half of all residual returns remain positive. What changes is the overlap. Exactly $(\frac{1}{2} + \alpha) \cdot N$ stocks end up with a position on the correct side of the residual return, matching the accuracy definition in Equations (5a) and (5b).

The realized profit follows by direct calculation

$$\pi_t = \sum_{n=1}^{N} \mathit{Position}_{n,t} \cdot \mathit{Resid}_{n,t} \tag{C.32a}$$

$$= \$0.005 \cdot \sum_{n=1}^{N} \text{Sign}(\mathit{Position}_{n,t}) \cdot \text{Sign}(\mathit{Resid}_{n,t}) \tag{C.32b}$$

$$= \$0.01 \cdot N \cdot \alpha \tag{C.32c}$$

The sum in the second line counts the number of correct recommendations minus the number of incorrect recommendations, $(\frac{1}{2}+\alpha)\cdot N-(\frac{1}{2}-\alpha)\cdot N = 2\cdot N\cdot\alpha$.

The encryption step works directly on the signs. Each iteration encrypts the positions and the residual returns as vectors of ±1 integers, multiplies them slot by slot, and sums across slots. Let S_t denote the decrypted integer total, so that $\texttt{Dec}[\texttt{Enc}(\pi_t)] = \$0.005 \cdot S_t$ in dollar terms. The implied accuracy is

$$\alpha_t = \frac{\texttt{Dec}[\texttt{Enc}(\pi_t)]}{\$0.01 \times N} = \frac{S_t}{2 \cdot N} \tag{C.33}$$

BFV arithmetic is exact on integer plaintexts, so $\alpha_t = \alpha$ in every iteration with no error whatsoever. This is why the left panel of Figure 2 reports $R^2 = 100.0\%$.

The 2% accuracy grid. The flip count above must be a whole number of stocks. Combined with the two balance conditions, this forces $\alpha \cdot N$ to be an even integer at every accuracy level. At $N = 500$, the levels $\alpha \in \{0.00, 0.02, \ldots, 0.16\}$ give $\alpha \cdot N \in \{0, 10, \ldots, 80\}$, all even. An odd-increment level like $\alpha = 0.03$ would require $\alpha \cdot N = 15$, which cannot be realized without breaking either the balanced book or the balanced returns. This is why Figure 2 moves in steps of two percentage points rather than one.

Continuous simulation. The right panel replaces the binary draws with normal ones, $\mathit{Position}_{n,t} \sim \text{Normal}(0, \$0.50)$ and $\mathit{Resid}_{n,t} \sim \text{Normal}(0, 2\%)$, and defines accuracy as the correlation, $\alpha = \text{Corr}(\mathit{Position}, \mathit{Resid})$. I want the scatter around the diagonal to isolate encryption error. A naive implementation would mix that error together with the sampling variability of a correlation

estimated from $N = 500$ draws. So the simulation pins every sample moment to its population target. Each iteration draws a Gaussian vector, demeans it, and rescales its sample standard deviation to $\sigma_P = \$0.50$. The result is the position vector. A second independent Gaussian vector is demeaned, projected orthogonal to the positions, and standardized to unit sample standard deviation. Call the result η. If we define $\sigma_R = 2\%$, then the residual returns are

$$Resid_{n,t} \;=\; \alpha \cdot \left(\tfrac{\sigma_R}{\sigma_P}\right) \cdot Position_{n,t} \;+\; \sqrt{1-\alpha^2} \cdot \sigma_R \cdot \eta_{n,t} \tag{C.34}$$

Direct algebra confirms that $\mathrm{Sd}(Resid) = \sigma_R$ and $\mathrm{Corr}(Position, Resid) = \alpha$. These relationships hold as sample identities rather than in expectation. Absent encryption, the profit is therefore exact in every iteration, $\pi_t = \alpha \cdot \sigma_P \cdot \sigma_R \cdot N$.

The two panels measure accuracy on different scales, and the parameters are chosen so that one recovery formula serves both. In the binary panel, the sample correlation between positions and residual returns equals $2 \cdot \alpha$, and each stock-level product has magnitude $\$0.50 \times 1\% = \$0.005$. In the continuous panel, the correlation equals α itself, and the products are governed by $\sigma_P \cdot \sigma_R = \$0.50 \times 2\% = \$0.01$. Doubling the residual scale exactly offsets halving the correlation convention. Both panels therefore divide the decrypted profit by the same $\$0.01 \times N$.

Integer quantization. BFV encrypts integers, and the continuous draws are real numbers. The simulation discretizes them following Hall, Fienberg, and Nardi (2011), scaling each one by a constant ($K_P = 100$ and $K_R = 500$) and then rounding

$$Position^{\star}_{n,t} \;=\; \mathrm{round}(K_P \cdot Position_{n,t}) \tag{C.35a}$$

$$Resid^{\star}_{n,t} \;=\; \mathrm{round}(K_R \cdot Resid_{n,t}) \tag{C.35b}$$

The scaled positions have sample standard deviation 50, and the scaled residual returns have sample standard deviation 10. So a three-sigma stock-level product has magnitude of roughly $150 \times 30 = 4{,}500$. The decrypted total at the highest accuracy level has expected magnitude $\alpha \times (K_P{\cdot}\sigma_P) \times (K_R{\cdot}\sigma_R) \times N = 0.16 \times (100{\cdot}\$0.50) \times (500{\cdot}2\%) \times 500 = 40{,}000$. This is far inside the ±516,000 plaintext bound. The recovered profit and implied accuracy are

$$\mathtt{Dec}[\mathtt{Enc}(\pi_t)] \;=\; \frac{S_t}{K_P \cdot K_R} \qquad \rightsquigarrow \qquad \alpha_t \;=\; \frac{S_t/(K_P{\cdot}K_R)}{\$0.01 \times N} \tag{C.36}$$

Results. The binary panel is exact. Every one of the $9{\times}100 = 900$ dots sits on the $y{=}x$ line, and the fitted R^2 is 100.0% with zero scatter at every accuracy level.

The continuous panel is nearly exact, with $R^2 = 99.9\%$. All of the continuous-panel scatter comes from the two rounding operations in Equation (C.35). None of it comes from the encryption itself, and none of it reflects sampling variability, which the exact-moment construction removes by design. Regenerating all the data behind Figure 2 takes about 3 minutes on a single desktop machine.

C.2 More Diagnostic Signal

Figure 3 in the main text compares two ways of judging an emerging manager, realized net profit and forecast accuracy. This appendix documents the Monte Carlo behind that figure and reports a companion test that does not appear in the main text. The companion test, shown in Figure C(1), compares two equally skilled managers who differ only in how they implement their ideas. Both figures come from a single simulation with 10,000 iterations. Each iteration follows $N = 500$ stocks for $T = 24$ months. No encryption appears anywhere in the exercise. It is a purely statistical comparison of two summary statistics. Encryption is what gives Alice access to the better one without forcing you to reveal your stock-picking rule.

Data-generating process. Returns follow Equation (21) in the main text. Each stock is described by 26 characteristics. The first 25 predict returns, and they do so symmetrically. The 26th plays no role in returns at all. Every characteristic evolves as an independent AR(1) process with innovations $\xi_{n,k,t} \overset{\text{IID}}{\sim} \text{Normal}(0, 1/\sqrt{25})$

$$\textit{Characteristic}_{n,k,t} \;=\; 0.9 \cdot \textit{Characteristic}_{n,k,t-1} \;+\; \sqrt{1-0.9^2} \cdot \xi_{n,k,t} \qquad \text{(C.37)}$$

The $\sqrt{1-0.9^2}$ scaling keeps each characteristic stationary, with the same marginal distribution in every month, $\textit{Characteristic}_{n,k,t} \sim \text{Normal}(0, 1/\sqrt{25})$. Because the 25 real characteristics are independent with variance $1/25$, their sum has unit variance. The 25% in Equation (21) is the population R^2 of the return equation. Each real characteristic has correlation $\sqrt{25\%/25} = 10\%$ with returns.

Persistence is what makes trading costly. At a month-to-month autocorrelation of 0.9, roughly a third to a half of the names in a manager's top decile rotate out each month, which generates a realistic rebalancing bill. Static characteristics would imply no turnover and hence no trading cost. IID characteristics would churn nearly the entire portfolio every month, and no manager would survive the commissions.

Three managers. Each manager, $m \in \{\text{You, Jim, Kat}\}$, observes a private signal, ranks all 500 stocks on it, and trades a market-neutral long/short portfolio.

You observe the first characteristic, $Signal^{\text{You}}_{n,t} = Characteristic_{n,1,t}$. Kat observes the third, $Signal^{\text{Kat}}_{n,t} = Characteristic_{n,3,t}$. The two signals are equally informative, with the same 10% population accuracy, but their realizations are independent. You and Kat pick different stocks in any given month even though you are equally skilled. Jim observes a corrupted signal

$$Signal^{\text{Jim}}_{n,t} \;=\; \tfrac{1}{2} \cdot Characteristic_{n,2,t} \;+\; \tfrac{\sqrt{3}}{2} \cdot Characteristic_{n,26,t} \tag{C.38}$$

The 26th characteristic is the spurious one. The squared weights sum to one, so Jim's signal has the same marginal distribution as yours. Only half of its variation loads on a real predictor, so Jim's population accuracy is 5%.

Accuracy in month t is the sample correlation between the manager's signal and the cross-section of realized returns

$$\alpha^m_t \;=\; \text{Corr}\big(\, Signal^m_{\cdot,t},\, Return_{\cdot,t} \,\big) \tag{C.39}$$

This is the same continuous accuracy convention as the right panel of Figure 2. Two properties of this definition drive everything below. First, accuracy is measured before costs. `EncryptIt` verifies the forecast, not the profit-and-loss statement that results from implementing it. Second, accuracy uses the entire cross-section, including the hundreds of names in the untraded middle. Net profit sees only the names a manager actually trades.

Trading and costs. A manager who trades L names per leg (L long and L short) holds the positions

$$Position^m_{n,t} \;=\; \begin{cases} 1/L & \text{if } Signal^m_{n,t} \text{ has rank } 1:L \\ -1/L & \text{if } Signal^m_{n,t} \text{ has rank } ([N-L]{+}1):N \\ 0 & \text{otherwise} \end{cases} \tag{C.40}$$

Total notional is one per side regardless of L, so a wider band just spreads the same capital more thinly. You and Jim trade $L = 50$ names per leg, the top and bottom deciles, putting 2% of capital in each name. Kat trades $L = 80$ names per leg, the top and bottom 16%, putting 1.25% in each name.

Each month the manager rebalances to the new signal's rankings. Gross profit is levered, and the manager pays a flat commission of 3.5bps of capital for every name whose position changed

$$gross^m_t \;=\; \Big(\textstyle\sum_{n=1}^{500} Position^m_{n,t} \cdot Return_{n,t} \Big) \Big/ \big(1{-}0.65\big) \tag{C.41a}$$

$$net^m_t \;=\; gross^m_t \;-\; 0.035\% \cdot \#\big\{\, n : Position^m_{n,t} \neq Position^m_{n,t-1} \,\big\} \tag{C.41b}$$

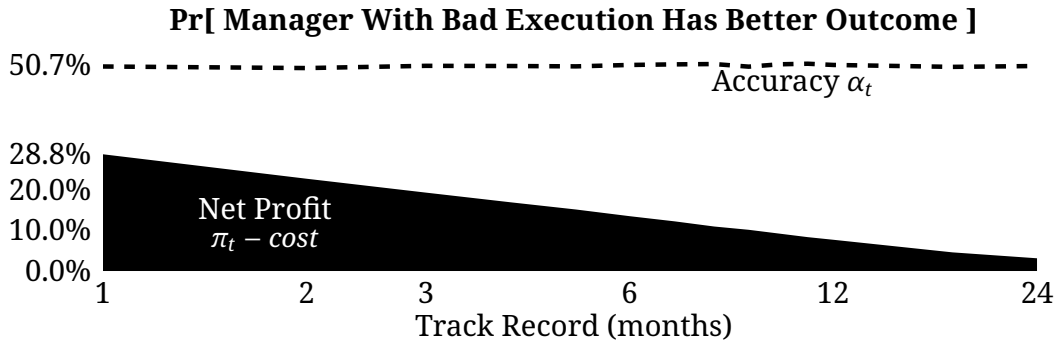

Figure C(1). Probability that Kat, an equally skilled manager with worse execution, has a better outcome than you as a function of track-record length (x-axis; months). Results across 10,000 simulations. Kat's stock-picking rule is exactly as accurate as yours, but she spreads her capital over the top/bottom 16% of stocks rather than the top/bottom 10%, generating higher turnover costs. The dashed line shows the fraction of simulations in which Kat's average forecast accuracy exceeds yours. The solid black region shows the same fraction for net profit. Accuracy correctly scores you and Kat as equally skilled. Net profit does not.

The commission itself is not levered. One accounting subtlety deserves mention. Each manager is picked up mid-career rather than started from cash. The simulation synthesizes a month-zero signal one AR(1) step before month one and derives starting positions from it. Month one is therefore charged a representative rebalancing cost rather than the full cost of building a portfolio from scratch, which would otherwise create an artificial ramp in cumulative profit over the first few months.

The commission was calibrated so that your net profit lands in the high single digits per year. Averaged across iterations at the full 24-month horizon, you earn 9.98% average accuracy and +9.63%/yr net profit. Kat earns 10.00% accuracy and −5.37%/yr net profit. Jim earns 5.01% accuracy and −8.29%/yr net profit. On gross profit the ranking tracks skill, 35.96% for you, 31.26% for Kat, and 18.05% for Jim. Kat's negative net profit is entirely the turnover cost of her wider band.

Beat rates. For each iteration i and each track-record length $h \in \{1, 2, \ldots, 12, 18, 24\}$, the simulation records each manager's average accuracy and annualized average net profit over the first h months. The plotted quantity is the fraction of iterations in which the inferior manager beats you

$$\Pr\big[\, m \text{ beats you at horizon } h \,\big] \;=\; \frac{1}{10{,}000} \cdot \sum_{i=1}^{10{,}000} \mathbb{1}\big[X_h^m(i) > X_h^{\text{You}}(i)\big] \qquad \text{(C.42)}$$

where X is either the accuracy average or the net-profit average. Figure 3 in the main text plots the beat rate for m = Jim. Figure C(1) plots m = Kat.

You versus Jim. Jim has half your stock-picking skill but identical execution. As a result, any difference between the two curves in Figure 3 reflects the noise floors of the two metrics. At every horizon, Jim's chance of beating you on net profit exceeds his chance of beating you on accuracy. The gap is 26.4% against 21.4% at one month, roughly 6% against 3% at twelve months, and roughly 2% against 1% at twenty-four months. Net profit filters your skill through fifty traded names, a leverage multiplier, and a stochastic commission bill. Accuracy reads the skill directly off all 500 forecasts. The noisier statistic hands the less-skilled manager more false wins at every track-record length.

You versus Kat. Kat has exactly your skill and worse execution. Her wider band hurts her twice. The names ranked 51 through 80 from the top are weaker picks than the names above them, so her per-name signal strength is diluted. And her 160 positions generate a larger monthly rebalancing bill than your 100. Neither handicap has anything to do with the quality of her forecasts.

Figure C(1) shows what this does to the two metrics. Kat's chance of beating you on accuracy sits at 50.7% at every horizon. Her signal and yours are equally informative, so each iteration is a coin flip, and no amount of additional track record changes that. Accuracy correctly scores the two of you as tied. Her chance of beating you on net profit tells a different story, starting at 28.8% after one month and falling toward 3% by twenty-four months. Judged on net profit, Kat looks progressively less skilled the longer she trades, even though her forecasts are exactly as good as yours.

The two figures make the argument in both directions. When managers really do differ in skill, accuracy identifies the better one faster than net profit does. When managers do not differ in skill, accuracy correctly reports a tie while net profit manufactures a difference out of an implementation choice. An allocator who sees only net profit would fund you and pass on Kat without ever learning that her problem is execution rather than forecasting. An allocator using `EncryptIt` can tell these two problems apart.

D Other Statistics

Online Appendix B shows how the notary can check whether the encrypted positions that you submit are appropriately sized and constitute a valid long/short portfolio. I now show how the same techniques can be used to test other properties of your portfolio: turnover, factor loading, and Sharpe ratio. All of these statistics can be added to `EncryptIt` by making a few small tweaks, or, in the case of the Sharpe ratio, no tweaks at all.

Turnover. A stock-picking rule can look profitable on paper but turn out to be too costly to run in practice due to high turnover. To measure turnover, you first compare each stock's current position to the position last period. Because $Position_{n,t} \in \{+\$0.50, -\$0.50\}$ in every period, the difference in positions can only take three values

$$\Delta Position_{n,t} = Position_{n,t} - Position_{n,t-1} \in \{-\$1.00, \$0.00, +\$1.00\} \tag{D.43}$$

Notice that, for this set of values, squaring and taking the absolute value are the same thing, $|\Delta Position_{n,t}| = (\Delta Position_{n,t})^2$. As a result, your portfolio's turnover will be proportional to

$$\text{Turnover}_t \;\propto\; \sum_{n=1}^{N} (\Delta Position_{n,t})^2 \tag{D.44}$$

This is a sum that the notary can easily compute with the encrypted data you would have given him anyways. He has $\{\texttt{Enc}(Position_{n,t-1})\}_{n=1}^{N}$ from the prior period, so the notary can compute the change in each stock's position as

$$\begin{aligned} \texttt{Enc}(\Delta Position_{n,t}) &= \texttt{Enc}(Position_{n,t} - Position_{n,t-1}) && \text{(D.45a)} \\ &= \texttt{Enc}(Position_{n,t}) \oplus [(-1) \otimes \texttt{Enc}(Position_{n,t-1})] && \text{(D.45b)} \end{aligned}$$

To compute $\texttt{Enc}(\text{Turnover}_t)$, the notary squares this quantity and sums across stocks. Before sending the result to Alice, he subtracts off a pre-specified cutoff, $\texttt{Enc}(\text{Turnover}_t - \text{Cutoff}) = \texttt{Enc}(\text{Turnover}_t) \oplus [(-1) \otimes \texttt{Enc}(\text{Cutoff})]$. All this is done using encrypted data, so the notary does not know whether your position in any stock has changed. But, when he sends the encrypted result to Alice, she can decrypt it and confirm the sign, $\texttt{Dec}[\texttt{Enc}(\text{Turnover}_t - \text{Cutoff})]$.

The calculations above require $\texttt{Enc}(Position_{n,t-1})$ and $\texttt{Enc}(Position_{n,t})$ to share the same key pair. Alice cannot redraw $(PublicKey, SecretKey)$ every period. The values used in Step #1 of Protocol 3 must persist across the whole evaluation window rather. This poses no real problem.

The choice of {−$1.00, $0.00, +$1.00} is convenient because it simplifies the algebra. But the intuition does not require unit scale. e.g., if your long/short positions were 4× as large, ±$2.00 vs. ±$0.50, then your portfolio's positions in each stock could swing 4× as far, {−$4.00, $0.00, +$4.00}. The notary knows how to adjust the turnover calculation because you have to privately commit to your position sizing at the start of the period.

Factor loadings. In the main text, I describe $Resid_{n,t}$ as the portion of the nth stock's return in period t that cannot be explained by the other managers Alice has already given money to. But there is nothing in the `EncryptIt` protocol which requires this particular origin story. Alice can send the notary other kinds of encrypted ciphertexts.

Suppose Alice is worried that you are simply doubling down on well-known risk factors. In that case, she might calculate each stock's "residual return" as

$$Resid_{n,t} \;=\; Return_{n,t} \;-\; \left(r_f + \sum_{k=1}^{K} \beta_{n,k} \cdot Factor_{k,t} \right) \tag{D.46}$$

r_f is the prevailing riskfree rate. $Factor_{k,t}$ is some aggregate risk factor, such as the excess return on the market, value, size, momentum, etc. $\beta_{n,k}$ is the nth stock's loading on the kth factor. If you are able to pick stocks even when handed these factor-adjusted returns, then Alice's original worry was not justified.

Sharpe ratio. Suppose you have used `EncryptIt` to demonstrate your skill to Alice each month for the past two years. In that case, Alice would now be in possession of a data set containing $T = 24$ observations about your long/short portfolio's excess return. At the end of each month, Alice decrypted a ciphertext to learn $\texttt{Dec}[\texttt{Enc}(\pi_t)] = \pi_t$. She has enough information to compute your Sharpe ratio without making any modifications to the baseline protocol

$$SR \;=\; \bar{\pi} \,/\, \bar{\sigma} \tag{D.47}$$

The formula above uses $\bar{\pi} = \frac{1}{24} \cdot \sum_{t=1}^{24} \pi_t$ and $\bar{\sigma} = \sqrt{\frac{1}{23} \cdot \sum_{t=1}^{24} (\pi_t - \bar{\pi})^2}$.

TANSTAAFL. There ain't no such thing as a free lunch. You do not want to reveal too many characteristics of your profitable new stock-picking rule to Alice. `EncryptIt` can be modified to prove each one in zero knowledge, which means Alice will learn nothing except the truth. But the truth is still informative. Recall the opening example from the introduction. There were $\binom{10}{5} = 252$ ways to take 5 long positions and 5 short positions in 10 stocks. Only 3 such long/short portfolios produced a return of +1.0%. A zero-knowledge proof ensures that Alice has no idea which of the 3 you held, but knowing that you were telling the truth eliminated 249 options.

References

Brakerski, Z. (2012). Fully homomorphic encryption without modulus switching from classical GapSVP. In *Annual Cryptology Conference*.

Brakerski, Z. and V. Vaikuntanathan (2011). Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Annual Cryptology Conference*.

Fan, J. and F. Vercauteren (2012). Somewhat practical fully homomorphic encryption. *Cryptology ePrint*.

Freivalds, R. (1979). Fast probabilistic algorithms. In *Mathematical Foundations of Computer Science 1979*.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. *ACM Symposium on Theory of Computing*.

Hall, R., S. Fienberg, and Y. Nardi (2011). Secure multiple linear regression based on homomorphic encryption. *Journal of Official Statistics*.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*.

Schwartz, J. T. (1980). Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*.

van Dijk, M., C. Gentry, S. Halevi, and V. Vaikuntanathan (2010). Fully homomorphic encryption over the integers. In *International conference on the theory and applications of cryptographic techniques*.